

UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Remindio
ferramenta de organização pessoal

Daniel Oliveira Sanches Leal, Patrick Silva
Souza

MONOGRAFIA FINAL
MAC 499 — TRABALHO DE
FORMATURA SUPERVISIONADO

Supervisor: Paulo Meirelles

São Paulo
2023

*O conteúdo deste trabalho é publicado sob a licença CC BY 4.0
(Creative Commons Attribution 4.0 International License)*

*Eu, Daniel, gostaria de dedicar este trabalho à minha esposa, Bruna, pela parceria, suporte, carinho e compa-
nheirismo todos esses anos durante essa jornada. Também
dedico aos meus pais e irmão por terem me possibilitado
trilhar esse caminho e, finalmente, aos meus tios, Regina
e Otávio, que possibilitaram que essa etapa se iniciasse.
Eu, Patrick, dedico este trabalho à minha amada por per-
manecer ao meu lado, por me apoiar, me compreender e
por me tornar uma pessoa melhor a cada dia. Também
preciso mencionar a minha família e os meus amigos que
me apoiaram, me acolheram e me orientaram durante a
jornada de graduação. Destaco a importância dos meus
primos Alessandro, André e Helloar para o ingresso na
universidade. Por fim, dedicação especial ao meu pai, Ail-
ton, por me acompanhar e por acreditar em mim sempre.*

Agradecimentos

Agradecemos ao professor Paulo Meirelles pela excelente orientação ao longo do processo de desenvolvimento, sempre nos munindo de conhecimento e de confiança. Agradecimentos a todos que contribuíram para a melhora do projeto, dedicando atenção e tempo para testar e sugerir correções e recursos. Por fim, agradecemos um ao outro pela parceria, pela amizade e pelo comprometimento com o projeto.

Resumo

Daniel Oliveira Sanches Leal, Patrick Silva Souza. **Remindio: ferramenta de organização pessoal.** Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

Na era da informação, a alta disponibilidade de entretenimento faz com que seu consumo dispute espaço com a dedicação às obrigações. Dessa forma, gerenciar tempo e energia tem se tornado um desafio cada vez maior. Nesse contexto, cresce a busca por soluções que propõem formas inteligentes e eficazes de otimizar o uso desses recursos no cotidiano. Dentre as possíveis soluções, destacam-se as tecnológicas, em forma de *sites* da web e de aplicativos para dispositivos móveis voltados para organização, ainda que poucas se dediquem por completo ao indivíduo. O presente documento descreve o processo de concepção de um *site* – construído com emprego de boas práticas de engenharia de *software* – voltado à organização individual assim como discute os tópicos de engenharia de *software* envolvidos no desenvolvimento do mesmo. O texto aborda as diferentes etapas acerca do desenvolvimento de uma plataforma do tipo, começando pela elaboração da proposta, passando pelas decisões tecnológicas envolvidas, pelo ciclo de desenvolvimento até a apresentação do *site* resultante e dos relatos que comprovam a aceitação dos usuários. O projeto culmina em um *site* bem recebido pelo público e disponível gratuitamente em <https://remindio.app> até a presente data.

Palavras-chave: Organização. Tempo. Desenvolvimento. Software. Site.

Abstract

Daniel Oliveira Sanches Leal, Patrick Silva Souza. **Remindio: personal organization tool.** Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

In the information age, the high availability of entertainment competes for attention with the dedication to responsibilities. Consequently, managing time and energy has become an increasingly significant challenge. In this context, there is a growing demand for solutions that propose intelligent and effective ways to optimize the use of these resources in daily life. Among the possible solutions, technological ones stand out in the form of websites and applications focused on organization, although few are entirely dedicated to the individual. This document describes the conceiving process of a website – built using good software engineering practices -- aimed at individual organization. It also discusses the software engineering topics involved in this process. The text covers different stages in the development of such software, starting from the proposal, going through the technical decisions and the development process, leading to the resulting website and user acceptance reports. The project culminates in a platform, in general, well-received by the public and available for free at <https://remindio.app> until the present date.

Keywords: Organization. Time. Development. Software. Site.

Listas de abreviaturas

HTTP	Protocolo de Transferência de Hipertexto (<i>Hypertext Transfer Protocol</i>)
MVP	Produto mínimo viável (<i>Minimum viable product</i>)
SGBDR	Sistema de Gerenciamento de Banco de Dados Relacional
ACID	Atomicidade, Consistência, Isolamento e Durabilidade
HTML	Linguagem de Marcação de Texto (<i>HyperText Markup Language</i>)
CSS	Folhas de Estilo em Cascata (<i>Cascading Style Sheets</i>)
CI	Integração contínua (<i>Continuous integration</i>)
CD	Entrega contínua (<i>Continuous deployment</i>)

Listas de figuras

1	Protótipo de interface da plataforma	2
1.1	Captura de tela do site do Notion, expondo a capacidade de unificação da plataforma para times	6
1.2	Captura de tela do site do Trello, demonstrando o funcionamento base da plataforma	7
1.3	Captura de tela do site do Todoist, demonstrando o foco e as funcionalidades	8
1.4	Captura de tela do site do ClickUp, demonstrando o foco e as funcionalidades	8
1.5	Captura de tela do site do Any.Do, demonstrando o foco e as funcionalidades	9
2.1	Arquitetura cliente servidor	14
2.2	Arquitetura de microsserviços	15
2.3	Arquitetura monolítica	16
2.4	Exemplo genérico de uma arquitetura hexagonal	17
2.5	Módulos e organização do backend	18
2.6	Implementação da arquitetura no backend com alguns componentes para exemplificação	19
2.7	À esquerda, diagrama genérico descrevendo composição de componentes. À direita, diagrama representando fragmento da arquitetura da página principal do Remindio. A intensidade do tom de cinza representa a profundidade de um componente.	25
2.8	Arquitetura do <i>frontend</i> a nível de aplicação	26
2.9	Estrutura de pastas do código-fonte que representa o <i>frontend</i>	27
2.10	Organização dos arquivos que representam um componente do sistema	27
2.11	Visão geral do diretório <i>/pages</i> do projeto	29
2.12	Comandos básicos e interação entre Git e GitHub	30

2.13	Captura de tela do <i>software</i> de prototipação	32
3.1	Produto mínimo viável do projeto	34
3.2	Ambientes e notas	35
3.3	Tarefas no quadro Kanban	37
3.4	Exemplo de <i>Pull Request</i> do <i>GitHub</i>	38
3.5	Exemplo de discussão produzida por revisão de código	39
3.6	Ciclo de desenvolvimento completo	39
4.1	Imagen mostrando um ambiente com diversas notas	42
4.2	Imagen do mesmo ambiente, demonstrando a proporcionalidade das notas em uma tela menor	42
4.3	Tela de registro e login da plataforma	43
4.4	Captura de tela mostrando a aba de feedback que aparece ao clicar no ícone no canto inferior direito	44
4.5	Imagen demonstrando um usuário com diversas ambientes na parte superior	44
4.6	Captura de tela demonstrando o temporizador Pomodoro ao lado do menu de lateral onde pode ser acessado	46
4.7	Capturas de tela mostrando o funcionamento da plataforma em um dispositivo móvel	46
4.8	Captura de tela demonstrando o menu de compartilhamento de ambientes	47

Listas de tabelas

1.1	Tabela resumindo análise concorrência, comparando diferentes concorrentes em quatro aspectos julgados importantes.	10
2.1	Quantidade de <i>downloads</i> semanais das três principais bibliotecas de componentes de acordo com o site https://npmjs.com . Essa quantidade de <i>downloads</i> foi obtida no dia 23 de novembro de 2023.	23

Sumário

Introdução	1
1 Concepção do Remindio	5
1.1 Análise de Concorrência	5
1.2 Comparativo	10
2 Fundamentação Tecnológica e Decisões Arquiteturais	13
2.1 <i>Backend</i>	13
2.2 <i>Frontend</i>	22
2.3 Versionamento	30
2.4 Prototipação	31
3 Processo de desenvolvimento	33
3.1 Idealização e validação	33
3.2 <i>Roadmap</i> e funcionalidades	35
3.3 Ciclo de desenvolvimento	36
4 Resultados	41
4.1 Funcionalidades desenvolvidas	41
4.1.1 Interação com notas	41
4.1.2 Autenticação	43
4.1.3 Formulário de feedbacks	43
4.1.4 Multi-ambientes	44
4.1.5 Temporizador pomodoro	45
4.1.6 Responsividade para dispositivos móveis	45
4.1.7 Compartilhar ambientes em modo de leitura	47
4.2 Percepções e resultados	47
4.3 Lições aprendidas	48

5 Conclusão	51
--------------------	-----------

Referências	53
--------------------	-----------

Introdução

Tempo e energia são recursos fundamentais e que precisam ser utilizados com parcimônia no dia a dia para a manutenção do bem-estar. No entanto a gerência desses recursos é um desafio, por conta da quantidade de obrigações e de distrações que escala ao longo dos anos. Por um lado, há a necessidade de evoluir nos âmbitos profissional e educacional na intenção de conquistar melhores condições de vida. Por outro, há o desejo de se distanciar das obrigações, geralmente satisfeito pelo entretenimento instantâneo disponível através das telas de dispositivos inteligentes.

Se a dedicação às obrigações for excessiva, caminha-se em direção a problemas de saúde que se manifestam de maneira física e mental. **JOCA et al.** (2003) mostra que o estresse, decorrente de uma possível carga excessiva de trabalho, relaciona-se à depressão. Em contrapartida **CARVALHO FARIAS et al.** (2011) constata a materialização do estresse em forma fadiga e de dores no corpo. No caso contrário, em que a dedicação dá lugar à procrastinação, o desequilíbrio se manifesta em forma de uma pobre saúde mental, sobretudo na adolescência, de acordo com **BRANCO** (2015).

Enquanto diversos estudos conectam má gestão de tempo e problemas de saúde, outros demonstram o que a experiência ajuda a intuir: práticas organizacionais ajudam a restaurar o equilíbrio entre obrigação e lazer. O estudo de **BRITTON e TESSER** (1991) aponta que a aplicação de boas práticas de organização implicou em bons resultados acadêmicos. De maneira semelhante, **DUCKWORTH e SELIGMAN** (2005) sugerem que autodisciplina é condição necessária para o desenvolvimento intelectual.

Nesse contexto se propõe o Remindio como ferramenta de organização pessoal em forma de um site da *web*, construído com boas práticas de engenharia de *software* em mente. O projeto é detalhado ao longo deste texto, bem como as tecnologias empregadas e o processo de desenvolvimento.

Objetivos do projeto

O Remindio é uma plataforma *web* de organização pessoal desenvolvida com o intuito de suprir as necessidades dos autores e que os mesmos enxergam nas pessoas em seu entorno. A solução foi concebida utilizando tecnologias sólidas e boas práticas na intenção de garantir estabilidade e desempenho.

O projeto surge da necessidade de uma maneira eficaz de se organizar, para evitar os problemas e alcançar as vantagens supracitadas. Apesar de o cenário atual conter uma

gama de soluções que podem ser utilizadas para a organização pessoal, enxergou-se espaço para a implementação de uma ferramenta completamente dedicada a tal propósito.

O usuário se organiza na plataforma através do sistema de notas e de ambientes, que constituem a sua principal funcionalidade: um usuário possui ambientes, e cada ambiente pode conter notas. Cada nota tem título, conteúdo e pode ser arrastada e redimensionada. A [Figura 1](#) demonstra protótipo da interface da plataforma.

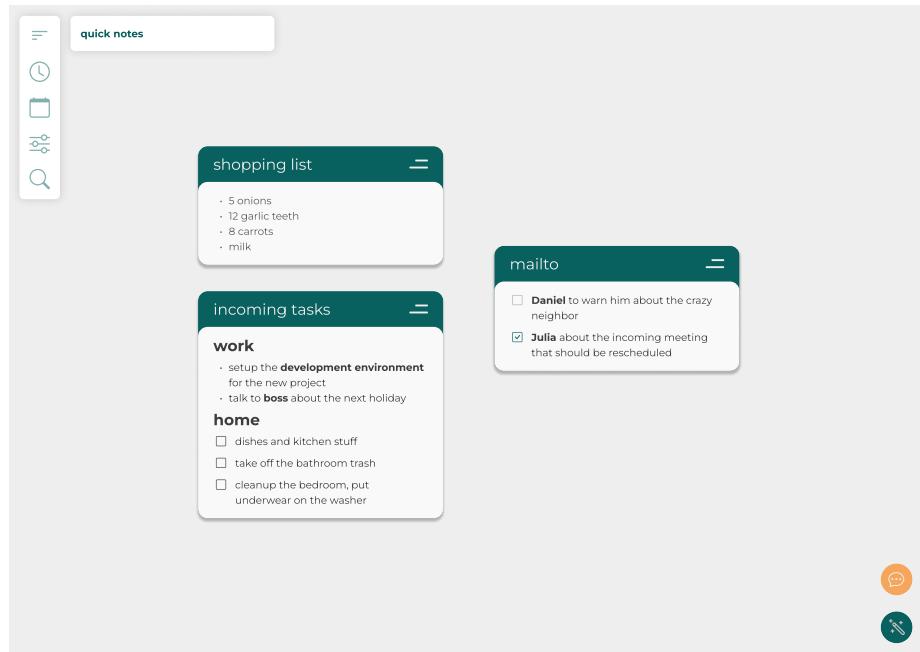


Figura 1: Protótipo de interface da plataforma

Acredita-se que a simplicidade e a flexibilidade inerentes do sistema descrito anteriormente abram espaço para que o usuário se organize com praticidade e clareza.

Além de ambientes e notas, o site conta com funcionalidades satélite para aprimorar a experiência do usuário.

- **O temporizador Pomodoro:** O usuário pode utilizar o temporizador para auxiliar na gerência de tempo. A técnica Pomodoro consiste em definir intervalos de tempo de foco e de pausa e alternar entre esses durante o dia. O autor da técnica destaca que o seu uso resulta em maior clareza de pensamento, maior atenção e facilidade em processo de aprendizagem
- **A agenda:** Calendário ao qual se pode associar lembretes. Aos lembretes, o usuário pode associar notas. É mais um artifício de gestão de tempo
- **Foco em notas:** Um conjunto selecionado de notas de um ambiente pode ser disposto de uma maneira organizada para que o usuário consiga focar sobre elas deixando as demais de lado. Pode ser útil quando um dado ambiente estiver com notas em excesso
- **Compartilhar ambientes:** Possibilidade de compartilhar ambientes e respectivas notas com outros usuários do site

Além da construção da ferramenta, o projeto também se propõe a compartilhar conhecimento do ponto de vista da engenharia de *software* em forma de tecnologias e desafios enfrentados durante o desenvolvimento.

Organização do trabalho

As seções a seguir discutem o processo de desenvolvimento e manutenção do projeto bem como analisam a evolução do mesmo da perspectiva dos usuários.

No [Capítulo 1](#), detalha-se a ideia do projeto e o seu processo de validação, através de uma análise de concorrência. O [Capítulo 2](#) apresenta as tecnologias utilizadas durante o desenvolvimento assim como justifica as escolhas feitas.

O [Capítulo 3](#) dissecava o fluxo contínuo de desenvolvimento adotado e seus elementos principais. No [Capítulo 4](#), há um balanço das funcionalidades desenvolvidas, além de uma análise de dados com base em: (1) formulário de avaliação disponibilizado no próprio site para os usuários (2) em números de acesso e (3) relatório de usuários ativos.

Capítulo 1

Concepção do Remindio

Como introduzido, a proposta desse trabalho é desenvolver, manter e incrementar o Remindio, uma plataforma de organização pessoal. A ideia da projeto surgiu:

1. da necessidade de aplicar conhecimentos à cerca de desenvolvimento de softwares para web em um projeto da vida real
2. da necessidade de uma plataforma dedicada para organização pessoal de um dos membros deste trabalho, apesar das diversas alternativas existentes no mercado

A insatisfação mediante as soluções já existentes surge do fato de que nenhuma delas supre a demanda de organização de um indivíduo em diversos contextos de sua vida: pessoal, estudos, trabalho, etc. As ferramentas populares acabavam se tornando complexas e incapazes de escalar, não atendendo diversos aspectos da organização pessoal além da gestão de tarefas e anotações. Aspectos como gestão de tempo, através de técnicas como a Pomodoro, e a existência de uma agenda, tudo em um único lugar, evitaria a necessidade de uma pessoa utilizar diferentes ferramentas para satisfazer suas necessidades. Finalmente, a ideia central do que veio a se tornar o Remindio parecia algo inovador, que remetia ao aspecto físico das notas de papel, só que de maneira digital, o que encaixaria para pessoas que estão maior parte do seu tempo em frente a um computador. Ademais, na concepção da ideia enxergou-se diversas oportunidades de funcionalidades que poderiam ser adicionadas ao funcionamento base da plataforma.

No entanto, seria necessário validar essas hipóteses, afim de garantir também se uma solução igual ao Remindio já não existia.

1.1 Análise de Concorrência

Um dos primeiros passos executados após a concepção da ideia foi a análise da concorrência. Para isso, foram analisados sistemas de *software* como: Notion, Trello, Pipefy, Todoist, Clickup e Any.do. Durante a análise, foi observado aspectos como funcionalidades existentes e quais problemas tentam resolver, aparência da plataforma e a sua usabilidade, como se interage com a mesma e o quanto distante em termos de concorrência essa ferramente estaria do Remindio.

Notion

Provavelmente o aplicativo de gestão mais popular atualmente. O Notion ganhou popularidade por ser extremamente robusto e extenso (A Figura 1.1 ilustra a página inicial do seu site). Além disso, criou um ecossistema de templates, em que a própria comunidade compartilha páginas para serem utilizadas de acordo com a necessidade do usuário.

O funcionamento básico do Notion é por meio de páginas que podem conter uma infinitude de sub-páginas. Todas as páginas utilizam um editor de Markdown, ademais pode-se inserir imagens, tabelas, bancos de dados, etc.

No entanto, é perceptível que o apelo do aplicativo se tornou a gestão de equipes e o uso empresarial, visando unificar diversas ferramentas utilizadas em uma só para atender a maior parte possível das necessidades de uma empresa.



Figura 1.1: Captura de tela do site do Notion, expondo a capacidade de unificação da plataforma para times

Desta forma, o Notion foi caracterizado como uma concorrência direta do Remindio, mas não próxima. Isso porque o foco da solução não é a organização pessoal, apesar de possuir diversas funcionalidades que podem ser utilizadas para tal fim, o que trás uma diferenciação para a alternativa desenvolvida nesse projeto.

Trello

Outra plataforma popularmente conhecida é o Trello (veja a página inicial da plataforma na Figura 1.2), este se vende com uma “superferramenta de produtividade” elaborado para reunir as tarefas, colegas de equipe e ferramentas de trabalho. Dessa forma, imediatamente não se posiciona como uma solução para organização pessoal.

O funcionamento do Trello é de um clássico quadro Kanban com mais algumas funcionalidades. A organização da plataforma se da por meio de quadros. Tais quadros contêm cartões que se distribuem em colunas e que podem ser movidos entre as mesmas. Cada coluna tipicamente representa um estado enquanto que o cartão representa uma tarefa. Os quadros podem ser compartilhados entre as pessoas.

1.1 | ANÁLISE DE CONCORRÊNCIA

Em conclusão, é perceptível que o Trello é um concorrente distante do Remindio, já que não oferece recursos focados no indivíduo. Um usuário que gosta do *layout* de Kanban para organizar suas tarefas pessoais pode utilizar o Trello, mas possui outras alternativas mais ricas.

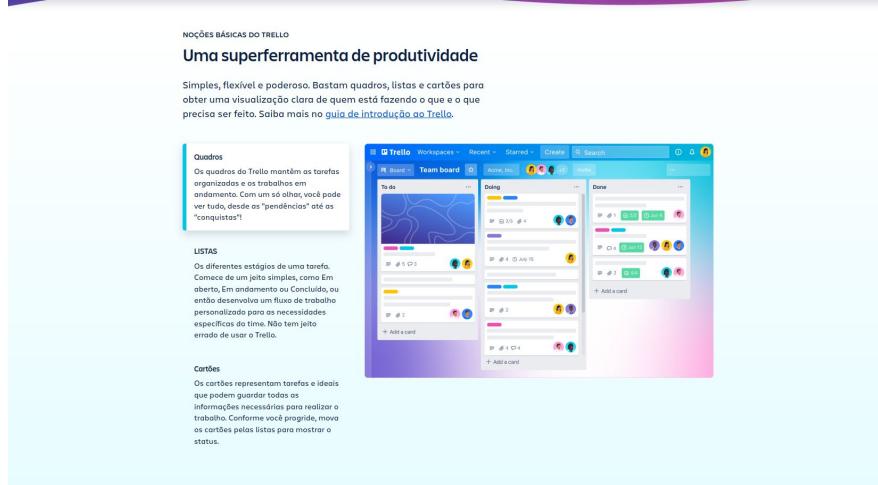


Figura 1.2: Captura de tela do site do Trello, demonstrando o funcionamento base da plataforma

Todoist

O Todoist é o concorrente direto do Remindio. A plataforma se vende como a ferramenta para organizar a vida e o trabalho, afim de atingir foco, organização e calma, como mostra a Figura 1.3.

A solução funciona através de uma lista de tarefas enriquecida. Essas tarefas podem ser organizadas em projetos e vão além do título, podendo conter descrições e subtarefas, além de etiquetas para auxiliar na organização, que por sua vez, podem conter níveis de prioridade. Outra funcionalidade disponível é o lembrete por data/hora, o que coloca a plataforma bem posicionada no âmbito da gestão de tempo.

A ferramenta oferece duas forma de visualização: através da lista e por meio do quadro Kanban. Além de diversas formas de agrupamento, que se integram com as funcionalidades de etiquetagem e data para oferecer mais flexibilidade ao usuário.

É perceptível o destaque e o foco da plataforma na gestão de tempo, trazendo diversas features e visualizações para atender esse aspecto da organização. Um dos pontos diferenciais é a já pré existência das visualizações por data de “entrega” de uma determinada tarefa.

No entanto, a interface da plataforma não é inovadora, possuindo o mesmo visual padrão de página com uma lista de tarefas. O que indica uma oportunidade de uma solução que possua uma forma de interação diferenciada como vantagem competitiva.

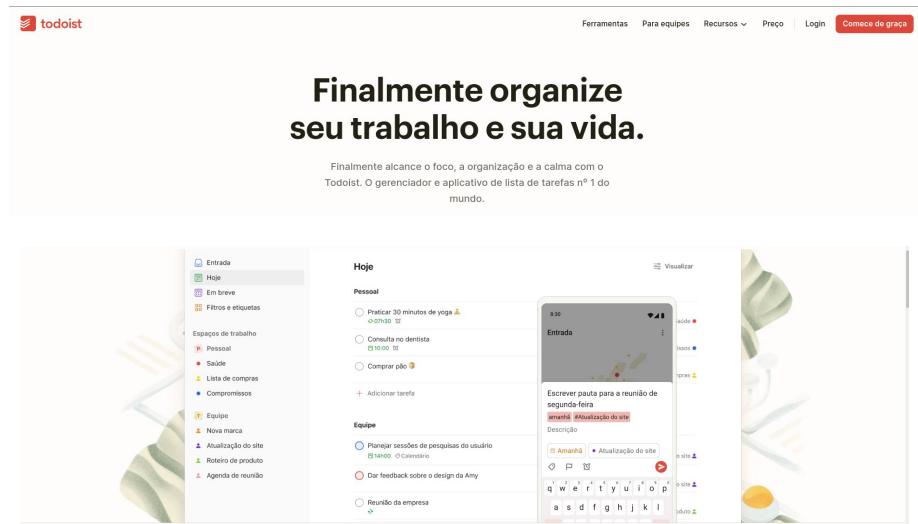


Figura 1.3: Captura de tela do site do Todoist, demonstrando o foco e as funcionalidades

ClickUp

O ClickUp é outra solução de organização, que no entanto também foca na gestão de times – no âmbito corporativo (a página inicial é ilustrada na [Figura 1.4](#)).

O app também se intitula como “a solução definitiva para substituir todas as outras”, e possui diversas funcionalidades que atendem casos de uso mais específicos como ambientes de cooperação para construção de fluxos e dashboards, novamente, focados no contexto de trabalho.

A solução funciona por meio de “espaços” quem podem ser visualizados de diversas formas, de maneira similar às outras plataformas. Estes podem possuir diversos projetos que é onde estão contidas as tarefas.

One app to replace them all.

All your work in one place:
Tasks Chat Goals Docs
Whiteboards Dashboards

Enter your work email

Get Started FREE FOREVER. NO CREDIT CARD.

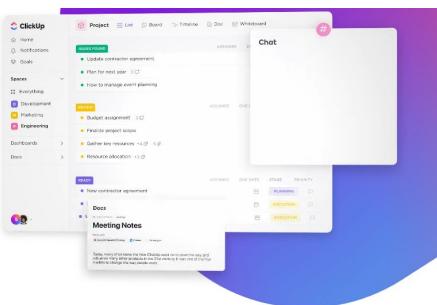


Figura 1.4: Captura de tela do site do ClickUp, demonstrando o foco e as funcionalidades

A plataforma apresentou a pior experiência inicial dentre as testadas. A diversidade de funcionalidades deixou a plataforma não só complexa como poluída e lenta, preju-

dicando a experiência geral. Nesse sentido, foi importante para a equipe observar que a curva de aprendizado de um usuário é um fator determinante para a adoção de uma ferramenta.

Devido aos pontos levantados, é perceptível a distância da plataforma em relação a concorrência com o Remindio.

Any.Do

O Any.Do (página inicial na [Figura 1.5](#)) apresenta-se como uma solução de organização que volta tanto para o indivíduo quanto para times.

A plataforma se anuncia com um clássico funcionamento por meio de um quadro Kanban, que o usuário consegue criar tarefas com descrições, tags e outras customizações e organiza-las em seções, também editáveis. Além disso, a solução oferece a visualização por listas, lembretes e integração com calendários.

One simple to do list for you and your team

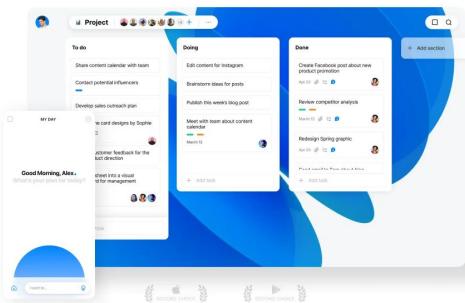


Figura 1.5: Captura de tela do site do Any.Do, demonstrando o foco e as funcionalidades

A ferramenta se assemelha ao já citado Todoist, caracterizando-a como uma concorrente direta do Remindio. No entanto, enquanto o Todoist tem interface simples e de fácil interação, o Any.Do apresenta interface pouco clara e levemente poluída

Todavia, essa proximidade a um concorrente é um ponto que não traz um diferencial a plataforma e portanto reforça a hipótese de que as soluções existentes são muito similares. Além disso, fica claro que existe espaço para diversas soluções cujo propósito é atender usuários buscando maneiras de se organizar, pois no fim o gosto do usuário desempenha um papel muito importante na escolha da ferramenta.

Síntese da comparação

A análise das plataformas concorrentes mencionadas mostrou que é possível classificar as mesmas levando-se em conta os seguintes aspectos:

1. **Objetivo:** Todos os concorrentes buscam resolver problemas ou possuem objetivos mais ou menos amplos
2. **Público-alvo:** Apesar de serem plataformas gratuitas, o público-alvo das mesmas é decorrente dos problemas que resolvem
3. **Interface:** A interface é um aspecto importante quando o assunto é organização e gestão de tempo. As plataformas mencionadas apresentam diferentes interfaces
4. **Inovação:** Representa um aspecto importante e determinante para o sucesso da plataforma. É esperado que o Remindio seja inovador, portanto é preciso que ele se destaque em pontos diferentes dos utilizados pelos seus concorrentes

A Tabela 1.1 sintetiza uma comparação das plataformas através dos aspectos listados acima e evidencia, por exemplo, que a maioria das soluções se volta para o público empresarial e busca inovar com um recurso ou um conjunto de recursos. Outro ponto importante de se mencionar são as interfaces: as plataformas, de modo geral, optam por dispor as informações em forma de páginas, listas e quadros Kanban.

	Objetivo	Público-alvo	Interface	Inovação
Notion	Espaço de trabalho conectado	Pessoas e empresas	Páginas	Robustez e extensão
Trello	Produtividade	Empresas	Listas, tabelas, quadros Kanban	Sistema de gestão de tempo completo
Todoist	Organização	Pessoas	Páginas	Visualizações por data de entrega
ClickUp	Gestão de times	Empresas	Espaços, projetos e tarefas	_____
Any.Do	Organização	Pessoas e empresas	Quadros Kanban	_____

Tabela 1.1: Tabela resumindo análise concorrência, comparando diferentes concorrentes em quatro aspectos julgados importantes.

1.2 Comparativo

A análise do setor mostrou dois principais pontos:

1. Existe espaço para diversas soluções
2. Não existe uma solução parecida com o Remindio

A primeira conclusão deve-se ao fato de que a análise provavelmente nem se debruçou sobre todas as ferramentas existentes e portanto existe espaço para diversas plataformas. A forma como indivíduos se organizam é algo muito pessoal. A simples constatação de que ainda existem pessoas que não utilizam ferramentas digitais e se organizam por meio de quadros, agendas, cadernos e etc. é, por si só, evidência da magnitude das possibilidades.

Além disso, apesar de algumas ferramentas serem claramente superiores a outras em recursos, usabilidade e experiência, ainda assim existem usuários que preferem utilizar a opção “pior”, justamente pelo fato de que gosto e costume desempenham um papel muito importante nas nossas decisões, que nem sempre são racionais.

A segunda se trata de uma informação objetiva adquirida através da análise, o que reforça a hipótese de que o Remindio, idealmente, possui potencial para ser uma ferramenta adotada e utilizada por um público.

Diante dessas informações, identifica-se um possível diferencial do Remindio na sua concentração exclusiva no indivíduo, em contraste com as outras ferramentas analisadas, que tendem a priorizar a organização de equipes ou a distribuir o foco entre o indivíduo e seu grupo. Essa singularidade no foco pode permitir que a ferramenta atenda de maneira mais precisa e eficaz às necessidades de organização pessoal dos usuários, ao direcionar todos os esforços para otimizar a experiência individual, em vez de dividir a atenção entre dinâmicas individuais e coletivas,

Um diferencial incontestável da ferramente proposta, diante da análise setorial, é o design. Enquanto as ferramentas analisadas apresentam interfaces majoritariamente padronizadas, em formatos de páginas, listas ou quadros Kanban, o Remindio inova ao se basear em notas, evocando uma associação com os *post-its*. Essa abordagem faz uma ponte com o mundo físico, refletindo diretamente a segunda heurística de Nielsen sobre a correspondência entre o sistema e o mundo real, que ressalta a importância de interfaces que sejam intuitivas e reconhecíveis para os usuários NIELSEN (1994).

Finalmente, a última hipótese é que o Remindio irá se destacar por seu dinamismo e flexibilidade, já que diante do design planejado, promete oferecer ao usuário a liberdade de personalizar sua experiência de organização, ao contrário das outras plataformas, que limitam as opções de organização a formatos predefinido. O Remindio permitirá que os usuários arranjam e dimensionem suas notas de acordo com suas preferências pessoais, criando um ambiente verdadeiramente adaptado às suas necessidades específicas. Essa capacidade de personalização irá conferir ao Remindio uma vantagem distintiva, possibilitando que cada usuário crie um espaço de organização único, que não só atende, mas também reflete suas preferências e estilo.

Capítulo 2

Fundamentação Tecnológica e Decisões Arquiteturais

Todo projeto de engenharia de *software* passa por diversas etapas que precedem o desenvolvimento da solução. Uma das etapas mais importantes é a definição das tecnologias que serão empregadas para construção do sistema. Nesse etapa, os engenheiros levam em consideração pontos como: requisitos do projeto, experiência, desempenho, custo, suporte/popularidade, entre outros. Dessa forma, neste capítulo será abordado o racional por trás das decisões tomadas que resultaram nas escolhas tecnológicas do projeto, assim como quais foram estas.

Backend e Frontend

Partindo do princípio, os sistemas *web* de maneira geral são sistemas de *software* construídos na arquitetura cliente-servidor, uma estrutura de aplicação distribuída em que as responsabilidades são divididas entre os dois componentes. O primeiro é o requerente dos recursos, no caso da *web*, os navegadores, chamado no contexto de desenvolvimento como o *frontend*, já o segundo – o servidor – possui a função de prover os recursos solicitados através do protocolo HTTP, sendo este popularmente chamado de *backend*.

2.1 *Backend*

Definição

Um dos principais componentes de qualquer sistema de *software* é o comumente chamado *backend*. O *backend* é a parte do sistema com a qual o usuário não interage, em que se encontram as principais regras do negócio, isso é, quais informações são criadas e gerenciadas, quais processos são ativados, entre outros. Tais regras são definidas com a finalidade de atender necessidades internas ou dos usuários.

Um exemplo bem simples seria uma transação financeira num sistema bancário, o usuário interagindo com o *frontend* define o valor que deseja transacionar e o destinatário,

esse lado envia as informações para o servidor que a partir das informações:

- Verifica se a conta bancária do cliente possui fundos suficientes
- Debita a conta bancária do remetente
- Credita a conta bancária do destinatário

Essa sequência de passos seriam as regras de negócio envolvidas numa transação

Geralmente quando se fala de *backend*, está se referindo a um processo rodando num servidor *web*, ou seja, um computador, capaz de receber requisições via internet – sendo o protocolo mais utilizado para essa comunicação o *HTTP* – um banco de dados e todo o código existente entre essas duas pontas responsável por executar as regras definidas.

Como todo sistema exige manutenção, melhorias e incrementos, a arquitetura desempenha um fator importante para a vida útil de um sistema e é um tema que sempre foi discutido pelos principais especialistas da área de engenharia de software

Arquitetura

Diferentes escopos podem ser o foco ao se discutir arquitetura. Em um escopo mais abrangente e simplificado, um sistema *web* como o Remindio utiliza uma arquitetura cliente-servidor, exemplificado na [Figura 2.1](#), já que há a separação entre o cliente – o requerente dos recursos – papel executado pelos navegadores, parte essa comumente referida como frontend. E no outro lado o servidor, quem provê os recursos.

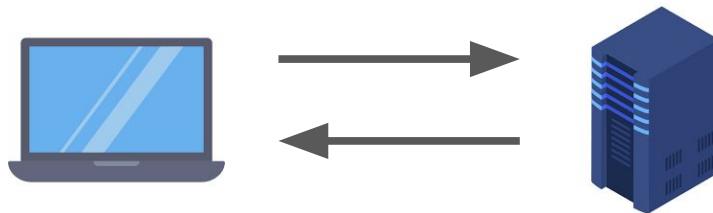


Figura 2.1: Arquitetura cliente servidor

Num escopo mais específico, relacionado ao sistema – focada em cada um desses elementos – a arquitetura discute a relação entre os componentes que compõe as partes. No contexto da parte interna (*backend*) uma discussão comum é entre a arquitetura de microsserviços *versus* uma arquitetura monolítica, sendo que ambas possuem pontos positivos e pontos negativos.

Na arquitetura de microsserviços, demonstrada na [Figura 2.2](#), as funcionalidades de um sistema estão separadas entre diversas aplicações, ou seja, bases de código distintas, cada uma responsável por um conjunto de funcionalidades. Pela natureza da arquitetura, como cada serviço é responsável por um conjunto de informações, as aplicações precisam se comunicar para que essas informações sejam trocadas. Se por um lado a troca de informações – não necessária numa arquitetura monolítica, já que tudo está contido em um mesmo lugar – traz uma complexidade, ganha-se independência no desenvolvimento e na implantação de funcionalidades, bem como na resiliência, pois em caso de falha, o sistema todo não é afetado.

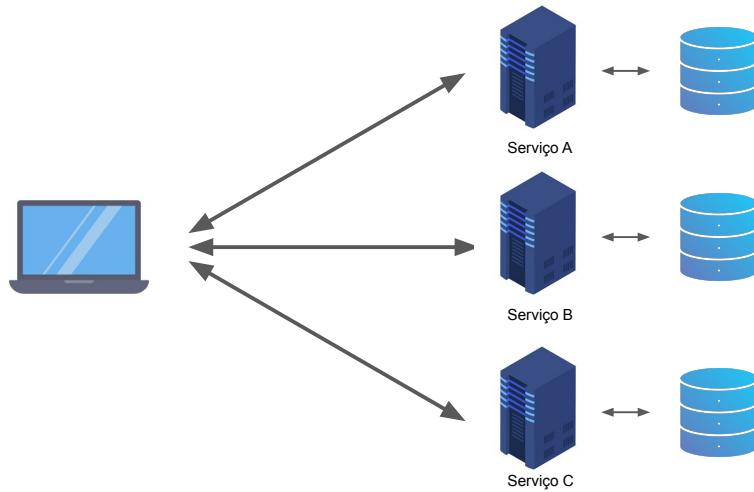


Figura 2.2: Arquitetura de microsserviços

Apesar das vantagens dos microsserviços, para o desenvolvimento do Remindio optou-se pela arquitetura monolítica, como pode ser vista na [Figura 2.3](#). A decisão entre essas duas possibilidades é amplamente debatida no mundo da Engenharia de Software. Os microsserviços viveram um *boom* de popularidade na última década e os monólitos foram inferiorizados, sendo culpados pelos problemas de performance, dificuldade de manutenção, instabilidade, entre outros. No entanto, nos últimos anos essa discussão tem sido revista dada a complexidade natural que os microsserviços exigem, assim, eles passaram a ser encarados como uma evolução do monólito e não mais como uma necessidade inicial de todo sistema, já que na opção mais simples há menos complexidade para desenvolver e implantar; é mais fácil de testar; possui um ciclo menor de desenvolvimento, portanto acelerando a entrega de features e valor para os usuários ([FOWLER, 2015](#)).

Ao não optar pela arquitetura de microsserviços para o Remindio, entendeu-se que os benefícios a serem colhidos com o monólito são superiores, já que o foco do trabalho é entregar valor e iterar sobre o *feedback* do usuário com mais agilidade. Em um monólito, não existe a preocupação quanto a comunicação entre as diversas funcionalidades de um sistema, como dito anteriormente. Somado a isso, não há também dificuldade em realizar testes *end-to-end*, já que uma funcionalidade não está possivelmente dividida entre aplicações diferentes. Por fim, os monólitos são mais fáceis de entregar em produção, exigindo menos recursos e menos sofisticação quanto a infraestrutura.

No entanto, afim de balancear aspectos técnicos de ponta, bem como aspectos de pro-



Figura 2.3: Arquitetura monolítica

duto, em contrapartida foi feito o investimento em uma arquitetura a nível de aplicação que permita usufruir de uma alta produtividade de código a medida que o projeto cresce.

Arquitetura do *backend*

Em um escopo mais específico ainda está a arquitetura da aplicação, que visa ditar a melhor forma de organizar os componentes, ou seja, interfaces, classes e a forma como é feita a comunicação dentre os diversos componentes a nível de código.

A arquitetura de um software é uma das principais preocupações de todo desenvolvedor. Saber estruturar uma aplicação e garantir essa estrutura à medida que ela cresce e envelhece, é de suma importância e esse assunto vem se popularizando nos últimos anos.

[COCKBURN \(2005\)](#) publicou um artigo propondo uma solução arquitetural para endereçar o problema de aplicações que crescem desordenadamente e entram em estados onde regras de aplicação e regras de negócio acabam se misturando e se espalhando por toda a base de código.

No livro *Clean Architecture* ([Robert Cecil MARTIN, 2017](#)), o autor expõe a sua visão sobre o assunto e mostra, a partir de sua vivência e experiência, que postergar ou tomar decisões ruins sobre a arquitetura do sistema de *software* tem um efeito devastador na produtividade dos desenvolvedores. Isto acaba prejudicando a empresa, que precisa da alta produtividade para publicar novas funcionalidades e se manter lucrativa.

Apesar da drástica diferença de idade e de particularidades que vêm de opiniões e visões diferentes, ambas as propostas possuem a mesma essência: separar o domínio da aplicação – as regras de negócio, aquilo que diferencia a aplicação, o motivo de sua existência e o problema que ela pretender resolver – das regras da aplicação, conjunto de regras que permitem o seu funcionamento, que possui importância, mas que não a diferencia das demais.

Para o desenvolvimento do Remindio, optou-se pela utilização da arquitetura Hexa-

gonal devido a estrutura clara que é proposta e a flexibilidade sobre a organização do domínio.

Arquitetura Hexagonal

Ao propor a arquitetura hexagonal, a intenção de Alistair (COCKBURN, 2005) era que as aplicações pudessem ser acionadas de diversas formas, por diversos clientes e serem testadas independente das tecnologias escolhidas. Isso porque tais tecnologias não deveriam ter importância para o domínio, sendo esse o ponto central da arquitetura. A Figura 2.4 exemplifica a arquitetura desenhada pelo autor.

Dessa forma, o autor propunha dois ambientes no ecossistema das aplicações: um *driver* e um *driven*. No primeiro, as tecnologias e formas que acionariam a aplicação, um servidor HTTP, uma linha de comando, etc. Do outro, as tecnologias a serviço da aplicação, utilizando a técnica da inversão de dependência afim de garantir a agnósticidade do domínio.

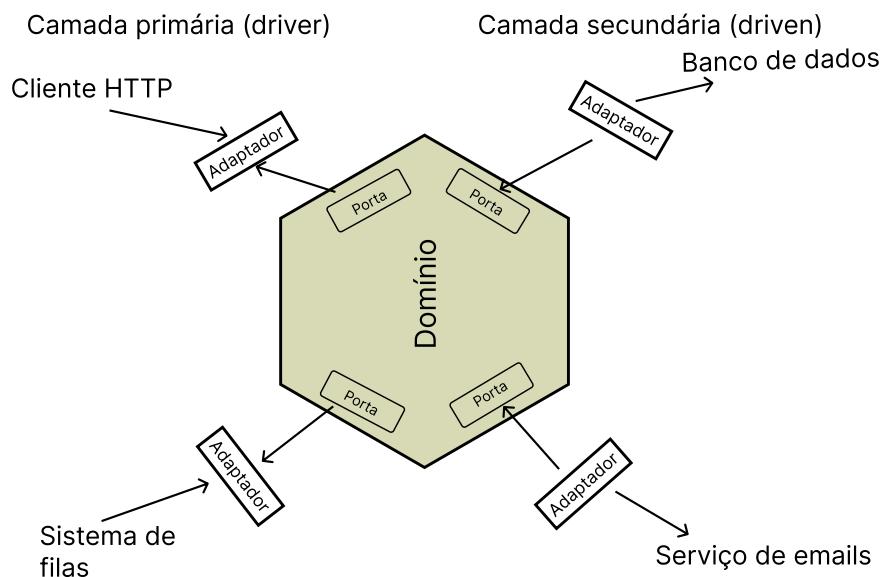


Figura 2.4: Exemplo genérico de uma arquitetura hexagonal

Para garantir essa independência entre as tecnologias e domínio, são utilizadas as portas (interfaces) e adaptadores. A função das portas é expor as funcionalidades realizadas pela aplicação, os casos de uso como explicita o autor – no caso das portas primárias (*driver*) – e os recursos demandados pelo domínio, para as secundárias (*driven*). Para atingir o isolamento ideal, as seguintes regras são fundamentais:

1. O domínio não pode conhecer detalhes das tecnologias
2. As tecnologias podem conhecer detalhes do domínio

A fim de atingir esses objetivos, são utilizados adapters GAMMA *et al.* (1994), um padrão de design de *software* concebido com a finalidade de converter informações que são

transferidas entre diferentes componentes, no caso, entre a camada do domínio e da tecnologia em questão e vice-versa.

Devido a isso, a arquitetura também é popularmente chamada de *Ports And Adapters*. Para as portas primárias, o próprio domínio implementa o contrato proposto. No caso das portas secundárias, cabe ao adaptador da tecnologia em questão cumprir o contrato, cuidando dos detalhes necessários para garantir o funcionamento. Um exemplo clássico é o da persistência de dados. A persistência é um ponto crucial de todo *software*, portanto ela é uma necessidade primordial, no entanto os detalhes dessa implementação não são importantes para o domínio. Dessa forma, o domínio da aplicação expõe um contrato para a persistência de um determinado objeto. Um adapter de uma tecnologia específica, como um banco de dados MySQL, implementa esse contrato e realiza toda a comunicação com o sistema de gestão de banco de dados relacional.

Para implementar a arquitetura, garantindo o cumprimento dos fundamentos exposto pelo autor, o *backend* do Remindio foi modularizado utilizando o Gradle – um sistema de automação de compilação e build – que permite configurar visibilidade entre os módulos da aplicação. A [Figura 2.5](#) demonstra a organização de pastas do *backend* e a [Figura 2.6](#) ilustra, com mais detalhes, o relacionamento entre alguns componentes presentes na implementação da arquitetura.



Figura 2.5: Módulos e organização do *backend*

Assim, podemos definir que os módulos *Postgres* e *http*, que fazem parte da área externa do hexágono conseguem ter acesso às classes e implementações do domínio. Da mesma maneira, podemos utilizar a funcionalidade do Gradle para limitar o visibilidade do domínio, impedindo que este conheça definições externas, garantindo a execução das regras da arquitetura.

Tecnologias de desenvolvimento

Para o desenvolvimento do *backend* de qualquer projeto de *software*, diversas decisões referentes as tecnologias adotadas são tomadas, visando balancear aspectos como experiência de desenvolvimento, frameworks disponíveis, bibliotecas existentes, etc.

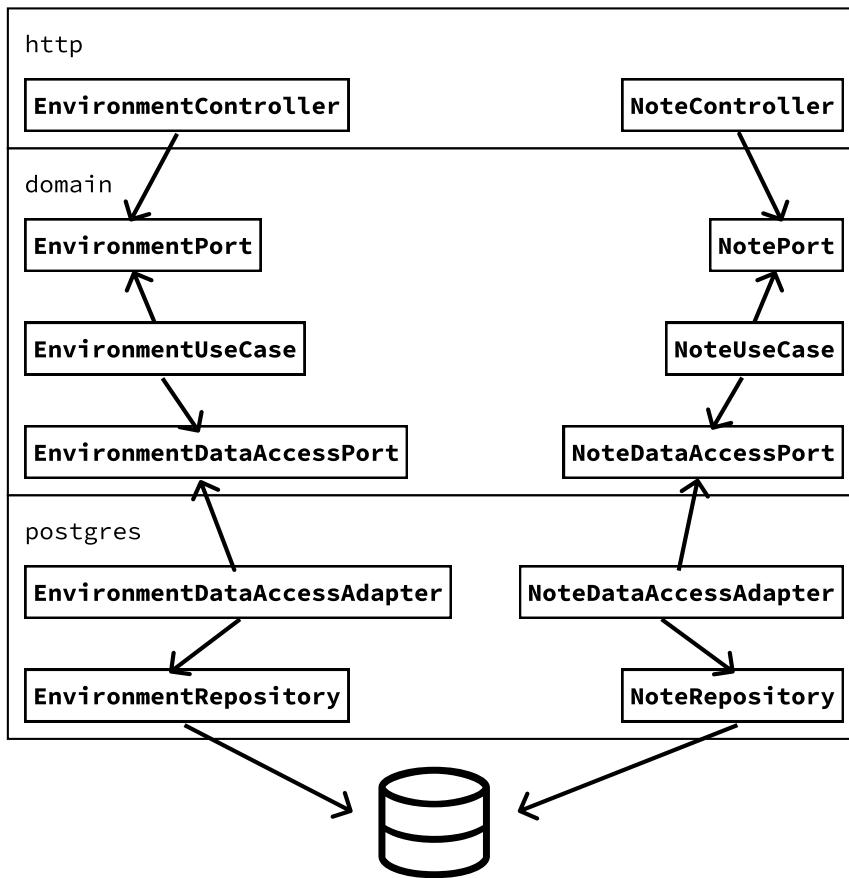


Figura 2.6: Implementação da arquitetura no backend com alguns componentes para exemplificação

Além das decisões já explicitadas como a arquitetura Hexagonal e monolítica, uma decisão muito importante é a linguagem a ser utilizada. Para o Remindio, na parte do servidor foi escolhida a linguagem Kotlin. Trata-se de uma linguagem moderna, orientada a objetos e funcional, fortemente tipada mas com inferência, que se tornou a linguagem oficial do Android em 2017, desde então ganhou cada vez mais popularidade. Além disso, a linguagem compila para a máquina virtual Java (JVM), o que permite que ela desfrute de um ecossistema extremamente robusto e popular, já que Java, além de ser uma linguagem popular, apesar de ter sido concebida na década de 90, ainda é umas das principais linguagens utilizadas no mundo corporativo, dada a confiança e credibilidade do ferramental que foi construído e mantido ao seu redor. Ademais, Kotlin possui interfaces, um componente que permite uma implementação fidedigna da arquitetura proposta para o projeto, trazendo flexibilidade e clareza, o que permitiu que o projeto se iniciasse sem a escolha de um banco de dados.

No referente às ferramentas, foi utilizado o *Spring Boot* e todo ferramental ao seu redor, afim de desfrutar de todo o rico ecossistema criado, mantido e melhorado para a JVM ao longo dos últimos anos. O Spring Boot é uma solução que utiliza da estratégia de convenção sobre configuração para o Spring, ou seja, disponibiliza diversas configuradas já feitas. Com isso, conseguimos subir um servidor web sem precisar definir diversas configurações, o que facilita o processo de desenvolvimento.

Como a arquitetura Hexagonal preza pela não utilização de ferramentas no domínio de sua aplicação, o uso do ferramental disponibilizado pelo Spring é mínimo nessa porção do código, limitando-se apenas para auxiliar no processo de injeção de dependência e abstrair a criação de transações com o banco de dados em contextos em que queremos garantir que dados sejam criados e atualizados de forma consistente.

Na camada primária, como dito anteriormente foi utilizado a solução web do Spring, que facilita ao prover um servidor HTTP e roteamento das requisições. Por fim, na camada secundária optou-se por utilizar o Spring Data JPA, uma biblioteca que abstrai e simplifica a comunicação com bancos de dados. O Spring Data implementa a especificação JPA, uma API padrão do Java que descreve uma interface comum para *frameworks* de persistência de dados, sendo o mais famoso o Hibernate.

Testes

Um aspecto muito importante de todo sistema é sua confiabilidade. Um *software* confiável garante uma melhor experiência para o usuário que o utiliza e para o desenvolvedor que o mantém, e a melhor maneira de garantir a confiabilidade de um sistema é por meio de testes automatizados.

Para o *backend* do Remindio, foi utilizado o *JUnit* e o *Spring Test* para teste unitários e de integração, respectivamente. Além desses *frameworks*, foi utilizado o *testcontainers* para criar um ambiente de testes completo a cada execução dos testes integrados.

O *testcontainers* é uma ferramenta que possibilita o lançamento de *containers*, um ambiente isolado que possibilita empacotar e rodar código, servidores e etc. de maneira uniforme e confiável. No caso do Remindio, é necessário um banco de dados para o correto funcionamento da aplicação, então, nos testes integrados em que se garante que as informações são salvas corretamente é necessário que um banco de dados esteja disponível e esta ferramenta foi utilizada afim de dispor esse requisito para execução dos testes.

Como dito anteriormente, os testes integrados são testes que validam o correto funcionamento de todo o fluxo de uma funcionalidade da aplicação, assim, é testado desde a requisição de um cliente via HTTP, até a conexão com o banco de dados e todas as regras de negócio existentes, como o envio de um e-mail, a consulta com algum serviço externo, etc.

No caso do testes unitários, estamos interessados em validar o correto funcionamento de uma regra. No exemplo dado no início deste capítulo, da transferência bancária, pode ser interessante escrever um teste unitário que verifica se o cliente possui a quantidade necessária de fundos para seguir com a operação. No Remindio, um exemplo de teste unitário implementado é referente a correta atualização de notas num ambiente, não resultando numa duplicação de dados, devido a natureza imutável de grande parte da linguagem utilizada.

Banco de dados

O banco de dados desempenha um importantíssimo papel em todo sistema, sendo portanto um componente vital para o funcionamento de toda aplicação. Ao longo dos

últimos anos, os bancos de dados passaram por grandes transformações, incluindo a forma como são encarados. Robert C MARTIN (2012) relata suas experiências ao longo das décadas demonstrando, nas empresas em que trabalhou, que além da evolução dessas tecnologias a forma em que a companhia em si e os desenvolvedores a tratavam mudou.

Nas décadas de 80 e 90, os bancos de dados eram encarados como a peça central de todo sistema e todo o design era pensado com essa tecnologia no centro. Nessa época, os grandes sistemas gerenciadores de bancos de dados estavam se popularizando. Essas soluções eram vendidas para corporações como a melhor e única solução para os problemas, isso levou os SGBDs a serem tratados não só como *softwares* capazes de desempenhar extremamente bem a função de guardar informações, mas como sistemas capazes de conter todas as regras de negócio e assim, operacionalizar todas as tarefas necessárias de uma empresa.

Nesse contexto, Martin relata os diversos problemas que surgiram por conta dessas práticas e assim passou a defender uma visão de que o banco de dados deve ser encarado apenas como uma tecnologia, e portanto, deve poder ser trocado facilmente caso exista uma opção melhor. Com essa proposição, a visão de Martin, e que foi compartilhada no desenvolvimento do Remindio, o autor recomenda postergar o máximo possível a decisão de qual banco de dados utilizar.

Essa nova perspectiva proposta tem os seus principais fundamentos nas melhores práticas aprendidas pelo especialista durante sua longa experiência. Regras de negócio contidas em *triggers* e *stored procedures* são mais difíceis de se manter e testar, as linguagens de *software* modernas passaram a se preocupar bastante com a experiência do desenvolvedor e portanto produzem códigos que realizam as mesmas tarefas de forma muito mais clara e coesa. Além disso, novas tecnologias emergem a todo momento, ter um pedaço central de código acoplado a uma tecnologia pode implicar um altíssimo custo e trabalho imensurável caso uma portabilidade seja necessária. Diante de todos esses fatores, Robert explicita que os casos de uso de um sistema, ou seja, os problemas que um software pretende resolver devem ser as peças centrais do design – o domínio como coração do *software*, como discutido no capítulo sobre Arquitetura. Essa visão, defendida também na sua obra recente Clean Architecture é compartilhada com Cockburn e foi a filosofia seguida ao encarar o banco de dados no desenvolvimento do Remindio.

Portanto, o banco de dados no Remindio foi tratado como uma tecnologia capaz de executar bem a tarefa de armazenar dados. No início do processo de desenvolvimento do projeto a decisão de qual banco de dados utilizar foi postergada, visando aliviar a carga cognitiva de mais uma decisão e focar nos casos de uso da aplicação, tratando as regras de negócio como pontos centrais. Como mencionado anteriormente, a decisão arquitetural possibilitou o adiamento dessa decisão, já que nela o banco de dados é encarado como um tecnologia que vai implementar uma interface específica para o armazenamento dos dados gerados pela aplicação. Durante o processo de desenvolvimento, teste local e até a primeira versão *alpha* entregue em produção foi utilizado o banco de dados H2, um SGBDR leve escrito em Java, em modo de memória, ou seja, os dados não eram persistidos em disco. Posteriormente, o banco de dados foi migrado para o PostgreSQL um SGBDR disponibilizado sob uma licença de *software* livre, mais robusto e amplamente utilizado.

A opção por um banco de dados relacional se deu pela simplicidade e confiabilidade

desses modelo, além do fato que um banco relacional é uma decisão segura e padrão para todos os tipos de aplicações que não tenham necessidades específicas, como é o caso do Remindio. As principais funcionalidades do sistema – ambientes e notas – se encaixam perfeitamente no modelo relacional já que a relação entre esses dois domínios é explícita. Além disso, esses bancos trazem as garantias ACID – acrônimo de *Atomicity, Consistency, Isolation, Durability* (Atomicidade, Consistência, Isolamento, Durabilidade) que, de acordo com [HAERDER e REUTER \(1983\)](#), são propriedades garantidas por bancos de dados relacionais e que fazem com que os dados armazenados estejam sempre válidos desde que as operações sejam encapsuladas em forma de transação – que são bem mais importantes para o projeto que a eficiência de bancos não-relacionais.

A decisão pelo PostgreSQL em específico se deu pela sua eficiência e garantias de isolamento, suporte nativo a UUID – que é acrônimo de *Universal Unique IDentifier* (Identificador Único Universal) modelo de identificação universal único utilizado pelas entidades do projeto –, o que garante maior eficiência na indexação das chaves primárias do que por exemplo o MySQL que não suporta esse modelo nativamente.

2.2 *Frontend*

O *frontend* corresponde à parte de um sistema *web* com a qual o usuário interage. A interação do usuário ocorre por meio do navegador que renderiza o site codificado. O *frontend* geralmente incorpora pouca ou nenhuma regra de negócio. Seu papel principal é obter os dados do usuário e enviá-los ao *backend* para que sejam apropriadamente processados e armazenados. Sendo assim, renderizar uma interface capaz de conduzir o usuário a todas as funcionalidades é papel indireto do *frontend* ([PAVLENKO et al., 2020](#)).

Pode-se resumir o papel do *frontend* no processo de cadastro de um usuário em

1. renderizar um formulário
2. exigir preenchimento correto de todos os campos
3. permitir submissão do formulário
4. enviar dados inseridos pelo usuário ao *backend*
5. exibir *feedback* ao usuário interpretando resposta

O termo *frontend* costuma ser utilizado para referir-se ao código-fonte responsável pela site da *web* ou para referir-se ao próprio site.

Linguagem

Enquanto existem diversas linguagens que possibilitam a implementação do *backend*, o *frontend* no quesito de desenvolvimento de sites da *web* se restringe a *JavaScript* e às suas variantes, a exemplo do *TypeScript*, que representa um superconjunto da primeira ao incorporar tipagem estática. *JavaScript* é a linguagem interpretada pelos navegadores modernos e que concede comportamento dinâmico às páginas *web*. Aliadas a *JavaScript*, estão *HTML* (de *HyperText Markup Language*) e *CSS* (de *Cascading Style Sheet*) que são linguagens de marcação e de folha de estilo, cujos propósitos são estruturar e estilizar o

site, respectivamente. Essas três linguagens são pré-requisito para desenvolvimento de sistemas da *web*, ainda que não sejam utilizadas diretamente em alguns cenários, que é o caso em que são substituídas pelo pré-processadores. No caso do *HTML*, pode-se citar *HAML* (de *HTML Abstraction Markup Language*), *Pug*, *Nunjucks*; para *CSS*, há *LESS* (de *Leaner Style Sheets*) e *SASS* (acrônimo de *Syntactically Awesome Stylesheets*). Todos esses exemplos representam pré-processadores que irão compilar o código-fonte fornecido para convertê-lo em um outro que seja válido para a respectiva linguagem.

Nesse contexto, no desenvolvimento do Remindio, utilizou-se *TypeScript* e *SASS*. *TypeScript* ganha popularidade nos últimos anos por representar um superconjunto de *JavaScript* incluindo tipagem estática, já que se trata de uma linguagem com tipagem dinâmica. O principal objetivo deste superconjunto é assistir o programador na manutenção de aplicações de larga escala (BIERMAN *et al.*, 2014). Enquanto isso, o propósito do *SASS* é estender o *CSS* adicionando recursos tais quais o aninhamento de seletores e os módulos (H. CATLIN e M. L. CATLIN, 2011). Esses recursos permitem redução de repetição de código e aprimoram a legibilidade do mesmo. Além da linguagem, outra decisão tecnológica importante durante a concepção de um projeto é a escolha do *framework* e das bibliotecas utilizadas. Isso é discutido a seguir.

Bibliotecas e *frameworks*

A variabilidade de tecnologias em um *site* da *web* está na variedade de bibliotecas e de *frameworks* que se pode utilizar. Existem inúmeras alternativas, mas destacam-se as bibliotecas de componentes e os *frameworks* baseados nas mesmas. Essas bibliotecas tornaram-se padrão de mercado durante a última década, como mostra a Tabela 2.1.

Biblioteca	Downloads semanais
<i>React.js</i>	18.366.574
<i>Vue.js</i>	3.520.047
<i>Angular.js</i>	2.660.078

Tabela 2.1: Quantidade de downloads semanais das três principais bibliotecas de componentes de acordo com o site <https://npmjs.com>. Essa quantidade de downloads foi obtida no dia 23 de novembro de 2023.

Diferentemente do que se leu na seção anterior, a arquitetura por trás de um *site* da *web* moderno costuma ser ditada pelo *framework* que é utilizado na fundação do mesmo. Esse costume restringe o papel do desenvolvedor, mas seus benefícios incluem aplicação natural de boas práticas de desenvolvimento de *software*, fácil implantação de ferramentas de teste e manutenção em sistemas de terceiros sem muitos esforços.

O Remindio se fundamenta no *framework* *Next.js* que, por sua vez, baseia-se na biblioteca *React.js*. As bibliotecas mencionadas na Tabela 2.1 mudaram a maneira de desenvolver sites da *web*, mas *React.js* se destaca como a mais adotada. As razões por trás da escolha de *Next.js* incluem:

- **grande e ativa comunidade:** a quantidade de *downloads* semanais da biblioteca, em comparação com as demais, demonstra que a mesma é mais popular dentre a comunidade de desenvolvedores

- **experiência e preferência do autor:** dada a limitação de tempo para implementação do projeto, considerou-se importante lidar com tecnologias atuais e de domínio dos desenvolvedores envolvidos
- **renderização do lado do servidor possibilitando otimização dos motores de busca:** sendo uma biblioteca de *JavaScript*, o código-fonte de um projeto baseado em *React.js* precisa ser completamente carregado pelo navegador antes de a página da *web* ser preenchida. Dessa maneira, os motores de busca, que processam o conteúdo da página para ranqueá-las nas pesquisas irão desqualificar a mesma. Esse é um problema que o *Next.js* resolve, no que implementa renderização do lado do servidor nativamente. A página da *web* é gerada do lado do servidor que hospeda o *site* e é enviada já construída ao navegador
- **tempo de carregamento inicial:** em *React.js*, como a renderização do conteúdo da página é feita via *JavaScript*, o navegador precisa carregar todo o conteúdo da página antes de renderizá-la. Dessa maneira, o tempo de carregamento inicial do projeto faz com que o usuário tenha que esperar, e essa espera é proporcional ao tamanho do projeto. O *Next.js* fornece ao navegador apenas o código-fonte responsável pela renderização da página requisitada nativamente, reduzindo o tempo de espera do usuário, tornando o projeto mais escalável

Para além de guiar o desenvolvimento de uma aplicação, o *framework* costuma expor uma série de utilidades que buscam resolver problemas comuns do desenvolvimento de um *site* da *web*. No entanto os projetos *frontend* costumam ter necessidades que vão além das funcionalidades incluídas no *framework* sobre o qual são construídos. Para a concepção do Remindio, as principais bibliotecas utilizadas foram as seguintes:

- ***Redux*** voltada para gerência do estado global da aplicação, mantendo organização dos dados gerados e obtidos do *backend*.
- ***Axios*** utilizada para comunicação eficiente com *backend* através da realização de requisições HTTP.
- ***Styled Components*** utilizada para estilização das páginas do *site*.
- ***Lexical*** utilizada para potencializar expressividade do conteúdo das notas, permitindo que usuário adicione listas ao conteúdo da nota e que formate o texto.
- ***Zod*** implementa validação e inferência estática de tipos para *TypeScript*, para garantir consistência enquanto os dados estiverem sendo veiculados na aplicação.
- ***FramerMotion*** proporciona uma série de componentes e utilidades para implementação de animações no *site*.

A subseção seguinte descreve como a linguagem, o *framework* e as bibliotecas mencionadas anteriormente compõem o projeto em termos de arquitetura.

Arquitetura do *frontend*

Como mencionado anteriormente, a arquitetura do projeto é determinada pela escolha do *framework* que o fundamenta. Neste caso, a escolha de *Next.js* culmina no uso de

arquitetura baseada em componentes, por conta da biblioteca *React.js*. A biblioteca foi divulgada ao público primeiramente em 2013 e, desde então, recebe destaque como sendo a mais utilizada para desenvolvimento de *sites* da *web*. Pode-se dizer que seu sucesso deve-se ao bom emprego da arquitetura baseada em componentes, que reduz a repetição de código e permite boa separação de responsabilidades.

De acordo com [A. W. BROWN \(2000\)](#), soluções baseadas em componentes têm sido a força motriz dos sistemas de *internet*, guiando o desenvolvimento das soluções em escala empresarial, pois são a concretização de uma das ferramentas intelectuais mais utilizadas para a resolução de problemas complexos de qualquer natureza: a decomposição de um problema em unidades menores e mais gerenciáveis.

Na arquitetura de componentes, cada componente encapsula uma funcionalidade bem definida e pode ser reutilizado tantas vezes quanto se queira na construção de um sistema maior. Além disso, os componentes podem ser compostos e podem se comunicar em diferentes direções.

Dentre os diversos benefícios da adoção da arquitetura em forma de componentes, pode-se mencionar:

1. **testabilidade:** testar um componente equivale a testar uma funcionalidade ou um conjunto de funcionalidades em forma de uma função ou uma classe
2. **manutenibilidade:** atualizações, correções, refatorações podem ser feitas com mínimo esforço
3. **reusabilidade:** componentes são produzidos tendo o reuso em vista

A [Figura 2.7](#) evidencia a aplicação natural da arquitetura baseada em componentes na construção de uma interface e exemplifica o uso da arquitetura no projeto em questão. Cada retângulo representa um componente. Componentes podem ter componentes-filho e a comunicação ocorre multi-direcional. Os componentes também podem se comunicar com seus ancestrais mais antigos e com componentes que não são ancestrais também. Dessa maneira se constrói uma árvore de componentes.

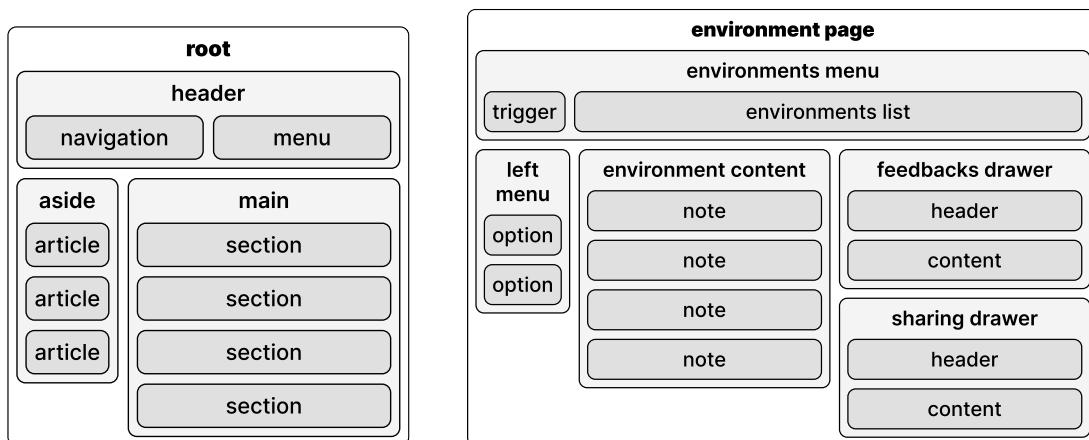


Figura 2.7: À esquerda, diagrama genérico descrevendo composição de componentes. À direita, diagrama representando fragmento da arquitetura da página principal do Remindio. A intensidade do tom de cinza representa a profundidade de um componente.

Apesar do projeto basear-se na arquitetura orientada a componentes, outros padrões de projeto são utilizados naturalmente na construção do mesmo, a exemplo do padrão *Observer*, que é intrínseco ao *JavaScript* e à maneira de interagir com os elementos da árvore de elementos que representa o *site*: Todos os elementos da página são controlados através de eventos. A esses eventos se associam observadores em forma de funções. Essas funções representam grande parte do comportamento dinâmico presentes nos *sites* modernos.

O projeto também emprega o sistema de gerenciamento de estado que compõe a arquitetura *Flux*, recentemente proposta pelo *Facebook*, ao utilizar a biblioteca *Redux* como mencionado na subseção anterior. Nessa arquitetura, o estado da aplicação é representado por armazéns (ou *stores*) cujas atualizações ocorrem através de ações (ou *actions*) que são enviadas ao expedidor (ou *dispatcher*): a única entidade capaz de atualizar os armazéns (BODUCH, 2016). As vantagens do uso dessa estratégia de gerência de estado incluem resolução natural de problemas de concorrência já que as ações são enviadas de maneira serializada e performance, pois somente as propriedades necessárias do estado são sobrescritas o que evita renderizações desnecessárias da interface.

De forma mais ampla, a [Figura 2.8](#) representa a interação entre os principais tipos de elementos que compõem a aplicação e também elucida o papel do *backend*, que é requisitado via protocolo HTTP. Os elementos mencionados na figura serão descritos em mais detalhe logo adiante, na subseção que descreve a organização do projeto. Tal organização deve receber atenção apropriada para garantir o bom funcionamento do projeto.

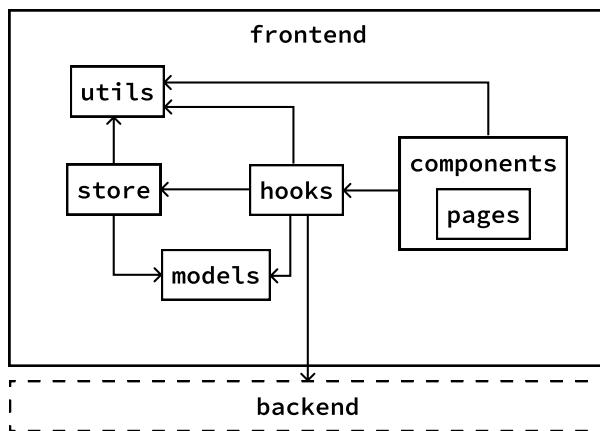


Figura 2.8: Arquitetura do frontend a nível de aplicação

Organização

Manter boa organização dos arquivos que compõem o código-fonte do projeto é essencial para que o desenvolvimento ocorra de maneira fluida. Para tanto, cada sub-diretório da raiz do código-fonte detém arquivos com responsabilidade definida. A [Figura 2.9](#) é um diagrama contendo os principais diretórios que compõem o repositório do *frontend*.



Figura 2.9: Estrutura de pastas do código-fonte que representa o frontend

Componentes

Um dos principais diretórios do projeto é o `/components`, pois detém os componentes que constituem a aplicação. O diretório `/components/app` contém componentes que representam os recursos da aplicação enquanto que o diretório `/components/common` contém componentes que são comumente utilizados por outros componentes. Esses diretórios contém subdiretórios, cada qual representando um componente. Dentro de cada subdiretório há o código-fonte do respectivo componente, que pode incluir componentes-filho, como ilustra a [Figura 2.10](#).

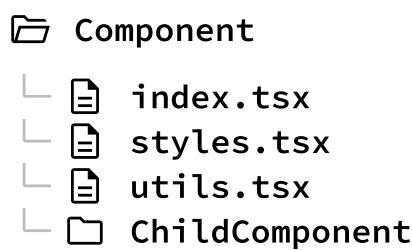


Figura 2.10: Organização dos arquivos que representam um componente do sistema

O arquivo `index.tsx` é o ponto de partida do componente, que une todos os outros arquivos e componentes-filho e que exporta o mesmo para uso em outros componentes. O arquivo `styles.tsx` contém a estilização do respectivo componente, utilizando a biblioteca *Styled Components*, para que o componente corresponda ao protótipo produzido. Por fim, o arquivo `utils.tsx` expõe funções e constantes que dizem respeito apenas ao componente e possivelmente a seus componentes-filho: seu papel é centralizar certo trecho de código para (1) torná-lo disponível para o componente e para seus componentes-filho; (2) para evitar

repetição de trechos; (3) para tornar o código mais legível e (4) para facilitar implementação de testes de unidade. A existência dos arquivos `styles.tsx` e `utils.tsx` depende da complexidade do componente, pois existem aqueles que são mais simples e não necessitam de estilização, pois baseiam-se apenas nos componentes criados em `components/common`, nos componentes nativos do `React.js` ou nos componentes de bibliotecas de terceiros. Apesar dos componentes serem o principal bloco de construção do projeto, eles dependem fortemente dos ganchos, descritos em seguida.

Os ganchos customizados

Em se tratando de bibliotecas de componentes, um dos principais desafios de seu uso é a manutenção da performance do projeto que pode se perder por conta de renderizações desnecessárias dos elementos que os componentes representam. Em `React.js`, essas renderizações acontecem, principalmente, por conta de atualizações de estado que ocorrem nos componentes e se propagam através da árvore de componentes. Essas atualizações de estado são necessárias, e podem ser solicitadas somente em componentes, através dos chamados ganchos (ou *hooks*) que são expostos pela biblioteca, ou em ganchos customizados.

Os ganchos (ou *hooks*) customizados também constituem parte importante do código-fonte e estão contidos no diretório `/hooks`. Assim como os componentes, eles encapsulam funcionalidades, mas sem necessidade de renderizar elementos. Eles se fazem necessários quando a funcionalidade em questão faz uso de ganchos `React`, ou de ganchos de bibliotecas de terceiros. No Remindio, os ganchos customizados podem ser categorizados em:

- **ganchos de acesso a informações**, que expõem propriedades do estado global da aplicação bem como funções para alterá-lo corretamente e para requisitar dados do *backend*. Um exemplo desse tipo é o que representa o estado de autenticação do *site*, e que expõe funções relacionadas a autenticação e propriedades de estado relacionadas ao usuário.
- **ganchos que implementam funcionalidades-chave do site**, expondo propriedades e funções que devem ser associadas a componentes para conferir-lhes tais funcionalidades. Um exemplo de gancho do tipo é o que permite que elementos sejam arrastados, admitindo uma referência para o componente que o representa e devolvendo o componente que representa o gatilho para o movimento de arraste já propriamente configurado.

No diretório `/hooks`, cada arquivo ou subdiretório representa um gancho. A estrutura de um gancho em forma de diretório é semelhante a estrutura de um componente.

Os modelos

O diretório `/models` contém arquivos que representam as principais entidades do sistema e entidades derivadas. Cada arquivo expõe uma série de objetos produzidos a partir da biblioteca `Zod` para validar os dados veiculados pela aplicação. Além de tais objetos, os seus respectivos tipos são definidos também.

A utilização combinada desses objetos de validação e dos tipos garante a confiabilidade de que o projeto precisa. Esses elementos são, portanto, largamente utilizados nos demais

arquivos que compõem o projeto.

As páginas

Uma das vantagens do emprego de *Next.js* no desenvolvimento do projeto é a funcionalidade nativa de roteamento baseado em arquivos. Por conta dessa funcionalidade, todo arquivo ou diretório contido em `/pages` é mapeado em uma página do site. A [Figura 2.11](#) descreve o conteúdo do diretório e mapeia cada arquivo contido no mesmo em uma rota do site. Por fim, vale mencionar que, do ponto de vista de implementação, as páginas são componentes.

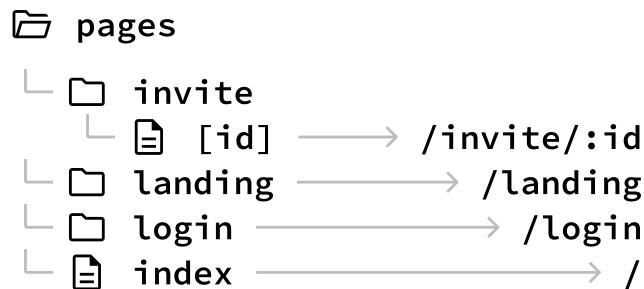


Figura 2.11: Visão geral do diretório `/pages` do projeto

Os armazéns

O diretório `/store` contém arquivos para estruturação dos armazéns que representam o estado global da aplicação. Esse estado pode ser alterado e é atualizado por toda aplicação. Para realizar a gerência apropriada de tal estado, o projeto faz uso da biblioteca *Redux*.

O diretório contém um arquivo para cada armazém e cada arquivo contém a definição do estado bem como a definição das ações que podem ser submetidas para a atualização do mesmo. Os diferentes armazéns são combinados e são disponibilizados para a aplicação com um todo.

Vale mencionar que a atualização do estado bem como o acesso ao mesmo são feitos apenas por intermédio dos ganchos customizados com o objetivo de padronizar e abstrair qualquer complexidade associada ao acesso, além de evitar repetição de código.

Testes

Em se tratando de *frontend*, pode-se dizer que os testes são ainda mais fundamentais para garantir integridade das funcionalidades ao longo e após o processo de desenvolvimento. Isso porque é preciso garantir uma variedade de comportamentos principais e secundários nas implementações.

Um dos principais *frameworks* de testes de unidade e integração para *JavaScript* é o *Jest*. Os testes são isolados por completo da aplicação e ele exige pouca ou nenhuma configuração para começar. Para realizar testes dos componentes *Next.js*, é preciso incluir

a biblioteca *React Testing Library* como dependência de desenvolvimento do projeto, já que esta expõe uma série de funções para renderizar os componentes e gerenciar seu estado apropriadamente. Com isso, para o Remindio, os autores deste trabalho optaram por cubrir as funcionalidades essenciais com testes de unidade.

Para as demais funcionalidades, dada a limitação de tempo hábil para cobertura completa do projeto, testes de interface foram implementados através da biblioteca *Cypress* que também é padrão de mercado neste quesito. Sendo mais simples de implementar, estes testes visam reproduzir os principais fluxos da aplicação, garantindo que estejam funcionando ao longo do desenvolvimento do projeto.

2.3 Versionamento

Uma prática padrão de qualquer projeto de *software* – independente do tamanho e complexidade – e usada no desenvolvimento do Remindio, é a utilização do Git para versionamento do código. O Git é um sistema de controle de versão distribuído que cria e mantém um histórico de um projeto a partir dos *Commits*, o comando que confirma a criação de uma nova versão a partir das alterações realizadas aos arquivos monitorados.

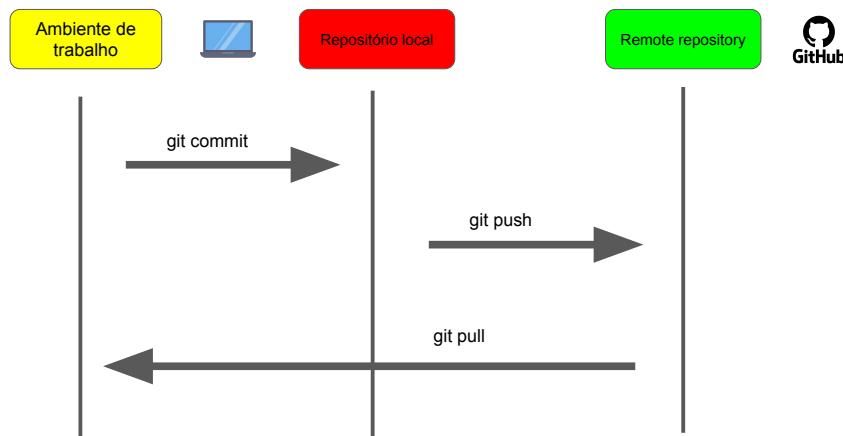


Figura 2.12: Comandos básicos e interação entre Git e GitHub

O *Git* também permite a criação de ramificações a partir de uma *branch base* – como a *main*, aquela que representa a versão estável atual do projeto, e muitas vezes o código em produção. Essas ramificações são criadas visando dar aos desenvolvedores um contexto isolado para a implementação de uma funcionalidade ou correção de um bug, que na finalização pode ser mesclada a branch principal.

Nesse contexto de versionamento, o *Git* é geralmente utilizado com uma plataforma que realize a hospedagem do código de forma remota, como o *GitHub* ou o *GitLab*, sendo a primeira utilizada no Remindio. Essas plataformas são necessárias para o compartilhamento do código entre os membros do projeto, permitir o acesso em qualquer ambiente de desenvolvimento, já que o código passa a estar em repositório que pode ser acessado

facilmente pelos membros, além de prover funcionalidades como os *Pull Requests* que desempenha um papel importante no ciclo de desenvolvimento do projeto pois permite que as implementações de um dos integrantes do trabalho possam ser revisadas pelo outro e vice-versa. Mais detalhes referentes ao processo de desenvolvimento do Remindio são detalhados no [Capítulo 3](#). A Figura 2.12 resume a interação entre os repositórios local e remoto além de apresentar os comandos básicos do *Git* para manter os dois repositórios atualizados.

2.4 Prototipação

Prototipação é um processo experimental executado geralmente por um ou mais *designers* no qual os mesmos implementam ideias para a construção de uma interface. O objetivo dessas ideias é solucionar um ou mais problemas de um tipo de usuário. É um processo que geralmente precede a etapa de desenvolvimento e que busca amadurecer a ideia de ferramenta e criar uma sólida fundação para o próprio desenvolvimento.

Os protótipos podem variar em termos de fidelidade com a interface final que o produto vai ter. Essa variação pode ser horizontal, no sentido de deter mais ou menos funcionalidades ou visualizações do produto final, ou vertical, quando o protótipo apresenta a interface com mais ou menos detalhes. Essa fidelidade depende do estágio de desenvolvimento, portanto do propósito do mesmo. Protótipos mais simples servem para amadurecer ideias e provar conceitos, por exemplo, enquanto os mais fidedignos servem para guiar os desenvolvedores. Durante essa etapa, os *designers* precisam ter em mente conceitos relacionados ao *design* de interface. É preciso pensar sobre o problema que a aplicação resolve e sobre os usuários da mesma. Em termos mais práticos, aspectos delicados são a organização geral da página, os meios de execução de ações, de obtenção de entrada do usuário, responsividade e a aparência ([TIDWELL, 2010](#)).

Sobre o processo de prototipação do Remindio, a elaboração de um *roadmap* do projeto foi um dos primeiros passos da concepção do mesmo. Dentre os objetivos do *roadmap*, destacam-se o amadurecimento da ideia, a preparação para o processo de desenvolvimento e a criação de uma maneira de apresentar o projeto. Apesar de nenhum membro ser especialista no processo de *design* de interface, ambos tiveram experiências no meio profissional. Isso é importante pois, de acordo com [TIDWELL \(2010\)](#), o comportamento dos usuários é previsível e existe um padrão. Dessa maneira, conceitos e ideias aplicadas no desenvolvimento de uma interface em um projeto podem ser utilizados em outras interfaces de outros projetos com o mesmo sucesso. Além disso, fazer parte do público-alvo da plataforma facilitou o trabalho.

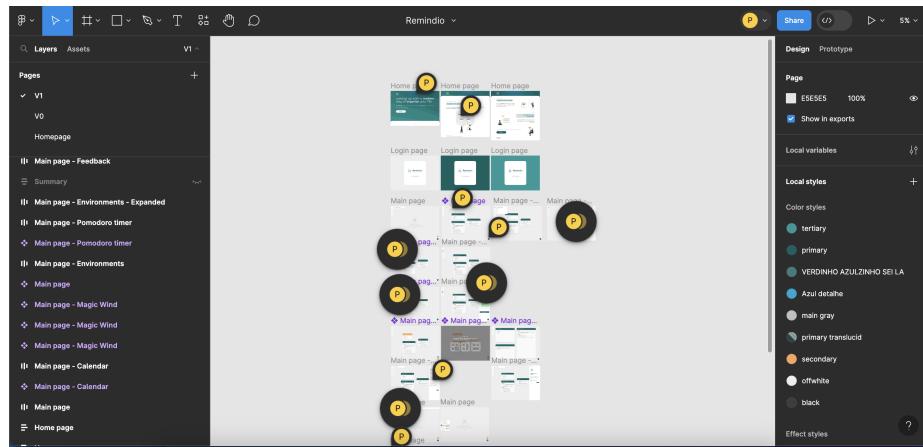


Figura 2.13: Captura de tela do software de prototipação

A Figura 2.13 representa uma captura de tela do *Figma*, que foi a plataforma utilizada para a prototipação. Apesar de existirem alternativas para a realização da tarefa, como o *Adobe XD* por exemplo, o *Figma* é considerada a solução mais popular e madura. Ainda na Figura, vale notar os círculos amarelos que representam comentários deixados por um dos membros da equipe contendo instruções para auxiliar no processo de desenvolvimento.

Ao longo do processo de desenvolvimento, nas reuniões semanais e durante o desenvolvimento das funcionalidades, o protótipo foi sendo modificado e aprimorado para corresponder à visão da equipe e às expectativas dos usuários. Além disso, com o protótipo encaminhado, a equipe pôde estabelecer os contratos de comunicação entre *frontend* e *backend* e pôde visualizar possíveis problemas.

Capítulo 3

Processo de desenvolvimento

No mundo corporativo, sistemas *softwares* são produzidos por times multidisciplinares, em que cada membro contribui com suas habilidades específicas e detém uma parcela de importância equiparada na materialização e manutenção do produto. Esta abordagem colaborativa, amplamente discutida em *Agile Software Development: The Cooperative Game* (COCKBURN, 2004), é não apenas catalisadora, mas imperativa para o sucesso em projetos de desenvolvimento de *software*, englobando a implementação de práticas e rituais que assegurem a execução do que foi planejado e priorizado.

As atuais práticas de desenvolvimento são produto de anos de evolução e experiência acumulada na indústria de *software*. Históricamente, sistemas eram desenvolvidos de maneira linear, de início a fim, baseados em requisitos anteriormente levantados pelos clientes. Contudo, esta estratégia herdada de outras áreas do conhecimento, frequentemente resultava em sistemas que nunca eram finalizados ou que excediam demasiadamente os prazos previstos, dada a ambiguidade das expectativas dos clientes ou a evolução de suas necessidades. Diante desse cenário, surgiram práticas revolucionárias, como as articuladas pelo Manifesto Ágil e exemplificadas em métodos como o Scrum e a Cultura DevOps, que transformaram radicalmente a condução dos processos de desenvolvimento de *software*.

Neste capítulo, aborda-se o processo de idealização e desenvolvimento acordado e executado durante o desenvolvimento do Remindio. Expõe-se quais abordagens e práticas foram adotadas visando diminuir a complexidade intrínseca do desenvolvimento de um *software*.

3.1 Idealização e validação

Todo sistema parte de uma ideia, muitas vezes concebida pela necessidade de uma solução para um problema, fornecendo uma resposta concreta e prática a desafios cotidianos. Esta perspectiva foi crucial para a gênese de plataformas globais renomadas, como *iFood*, *Uber* e *WhatsApp*, cujas soluções digitais abordam problemas específicos de logística alimentar, transporte urbano e comunicação interpessoal, respectivamente. O elo comum entre essas inovações é a transição de uma ideia inicial – muitas vezes em forma de um

mero conceito – até a materialização em uma solução tecnológica que ganha aceitação e adoção massiva.

Na fase inicial, a validação da ideia é um passo fundamental, que pode ser orquestrada de diversas formas, proporcionando um terreno sólido sobre o qual o projeto pode ser construído. Empregando um método que se baseia em teorias do *Design Thinking*, a validação pode ser feita por meio de uma série de iterações que procuram entender, definir, idealizar e prototipar a ideia em questão ([T. BROWN, 2009](#)). Nesse processo, a equipe de desenvolvimento se envolve em um ciclo contínuo de *feedback* com os usuários-alvo, garantindo que a solução proposta esteja alinhada com as necessidades e expectativas do público.

Outra metodologia que ganha espaço nesse contexto é a abordagem *Lean Startup*, proposta por [RIES \(2011\)](#), que enfatiza a importância de lançar rapidamente um Produto Mínimo Viável (MVP), para aprender o máximo possível sobre os usuários com o menor esforço possível. O MVP não é necessariamente a versão mais crua do produto, mas sim a forma mais simples da ideia que permite o aprendizado máximo sobre os clientes com o mínimo de esforço. E essa foi abordagem seguida no projeto, com um MVP demonstrado pela [Figura 3.1](#).

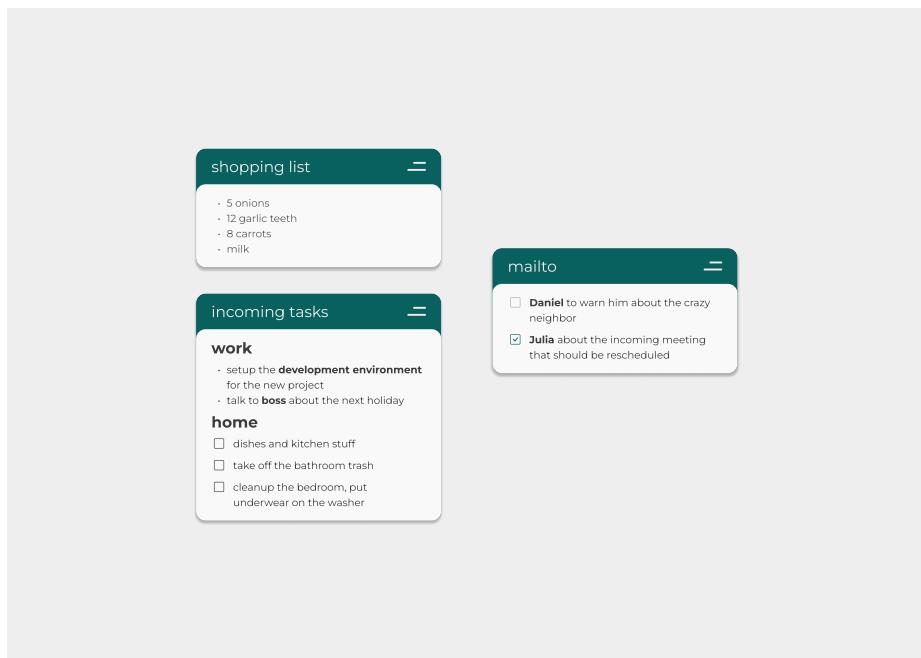


Figura 3.1: Produto mínimo viável do projeto

A decisão de desenvolver um Produto Mínimo Viável (MVP) foi embasada tanto nas competências da equipe do projeto quanto nas vantagens inerentes à abordagem, considerando o contexto e as teorias de *Design Thinking*, como a outra alternativa. Primeiramente, é válido ressaltar que a equipe já possuía experiência prévia no desenvolvimento de sistemas *web*, constituindo um alicerce sólido para a execução da iniciativa. Ademais, o Remindio foi deliberadamente projetado para estabelecer uma conexão palpável com o mundo real, alinhando-se à heurística de “Correspondência entre o sistema e o mundo real” ([NIELSEN, 1994](#)), uma vez que sua funcionalidade central simula o ato de afixar notas em um quadro.

Por último, a concepção do projeto não era inédita, pois uma versão mais incipiente já havia sido desenvolvida e avaliada positivamente em termos de aceitação da ideia.

3.2 Roadmap e funcionalidades

Anteriormente a concepção do MVP, um *roadmap* foi elaborado para o projeto. Um *roadmap* é uma prática estratégica utilizada para fornecer uma visão da possíveis funcionalidades e possibilidades a serem priorizadas e implementadas no futuro, trazendo maior clareza das oportunidades.

Para isso, uma versão preliminar dos fluxos e telas dessas funcionalidades foi elaborada no *Figma* e, a princípio, a seguinte rota ficou definida para o projeto:

1. Ambientes e notas, exemplificado na [Figura 3.2](#)
2. Agenda
3. Temporizador Pomodoro
4. Varinha mágica

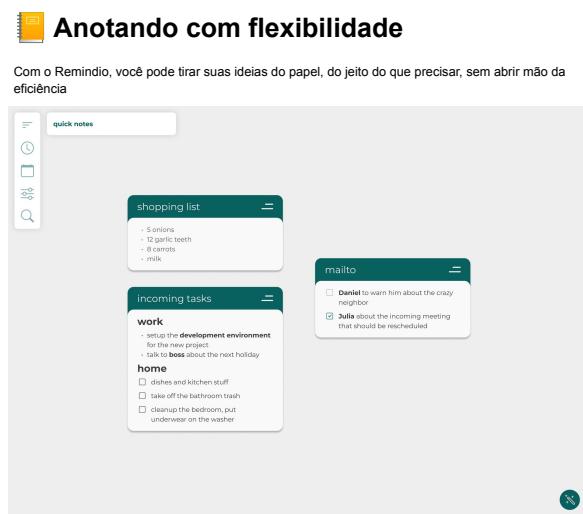


Figura 3.2: Ambientes e notas

Ao finalizar, já havia entendimento de que as funcionalidades não seriam priorizadas na ordem estipulada ou que seriam essas as *features* a serem desenvolvidas. A concepção do *roadmap* possibilitou uma visão das possibilidades e exercitou o uso da criatividade para pensar em soluções que se encaixam no contexto do projeto. Um exemplo disso foi a *Magic Wand*: uma ideia que surgiu durante o processo de prototipação, visando permitir que o usuário foque em somente algumas notas, deixando as outras do ambiente para trás, sem grande esforço e preservando ao fim a organização previamente estabelecida para o seu ambiente.

A fase subsequente envolveu a definição da versão inicial para ser implementada em produção, em busca de permitir a utilização prática pelos usuários — inicialmente, pelos próprios membros envolvidos no projeto. Manter código em produção, concomitantemente

com seu desenvolvimento e aprimoramento, constitui uma premissa neste trabalho, refletindo a realidade da engenharia de *software* contemporânea. Esta prática oferece benefícios cruciais, como *feedback* do usuário em tempo real e validação prática de funcionalidades e desempenho (HUMBLE e FARLEY, 2010). A utilização do sistema em produção permite um ciclo de *feedback* contínuo, fomentando ajustes rápidos e precisos e oferecendo um panorama realista sobre o comportamento e a performance do sistema de *software* em condições operacionais reais.

Para o Remindio, o MVP consistia em permitir interação do usuário com o ponto focal do projeto que são as notas em um ambiente. Nessa primeira versão, afim de simplificar a primeira entrega, os ambientes foram deixados de lado e o foco foram as notas, permitindo todas as interações planejadas: criar, editar, mover e redimensionar.

3.3 Ciclo de desenvolvimento

O ciclo de desenvolvimento adotado para o projeto Remindio foi sustentado por reuniões semanais, em que se

- discutia sobre o que foi feito na semana anterior
- planejava e discutia os pontos de atenção da semana subsequente

Durante a reunião, devido a celeridade pela existência de somente dois membros, cerimônias clássicas no mundo do Scrum como planejamento e refinamento eram realizadas. O planejamento envolve a discussão sobre a priorização das tarefas a serem refinadas, considerando a introdução de novas funcionalidades, melhorias identificadas ou correção de erros. A execução desta etapa permite um maior alinhamento das expectativas da equipe em relação às entregas. Um aspecto crucial é garantir que todos os membros estejam em harmonia quanto às prioridades. Como as equipes são compostas por membros de diferentes áreas, diversas visões e necessidades são compartilhadas, possibilitando que decisões sejam tomadas levando em consideração um contexto mais amplo, o que influencia positivamente na precisão das decisões tomadas.

Já o refinamento, que acontecia logo em seguida tinha como objetivo definir os pontos técnicos das tarefas, discutir a forma como seriam resolvidos os problemas e firmar os contratos de *APIs*, por exemplo. A realização desse processo permite que os indivíduos envolvidos compartilhem suas perspectivas sobre como um determinado problema pode ser solucionado. Dessa forma, a proposta é testada sob diferentes pontos de vista, possibilitando a consideração de vários cenários. Isso permite que potenciais problemas e dúvidas técnicas sejam antecipadas ou até mesmo assumidas, com o objetivo de acelerar o desenvolvimento, de forma que o time já esteja alinhado e acordado antes que o desenvolvimento das funcionalidades ou correção de *bugs* sejam iniciadas.

No contexto do projeto, o planejamento permitia que as duas frentes de desenvolvimento compartilhassem suas visões sobre as possibilidades do que poderia ser implementado, e como enxergavam as prioridades. Durante a reunião, as *features* a serem priorizadas eram revisitadas, visando possibilitar que estas fossem entregues com menos esforço ou de maneira faseada com pequenos incrementos, permitindo que a nova funcionalidade chegasse ao usuário mais rapidamente, em um estágio inicial, mas não necessariamente

completa. Esse tipo de entrega permite colher as impressões do usuário sobre a funcionalidade e viabiliza melhorias. Assim, o refinamento era realizado para esclarecer como seriam realizadas as comunicações entre o *frontend* e o *backend* e como ficaria a interação com o usuário.

A metodologia *Kanban* foi empregada para a gestão das atividades do projeto, objetivando proporcionar uma visibilidade das tarefas em execução, organizar de maneira estruturada as iniciativas futuras e manter um registro das potenciais melhorias, além de problemas identificados e um detalhamento apropriado de cada tarefa. Essa abordagem oferece uma visualização clara e acessível do fluxo de trabalho, permitindo que a equipe rastreie e priorize eficientemente as atividades, ao mesmo tempo que mantém uma documentação das operações em andamento e pendências. A Figura 3.3 ilustra o uso da técnica mencionada pelos autores deste trabalho durante o seu desenvolvimento.



Quadro de desenvolvimento

Backlog	Ready For Refine	Ready For Development	In progress
Mensagens de feedback ao usuário	+ New	Criar ambiente tutorial	Pomodoro back
Agenda		Pomodoro front	Daniel Oliveira Sanches Leal
Pensar na responsividade	+ New	Patrick Silva	Backend
+ New		Frontend	+ New

Figura 3.3: Tarefas no quadro Kanban

Após a implementação de cada tarefa, o código resultante era submetido para revisão pelos pares através dos *Pull Requests*, uma funcionalidade do *GitHub* utilizada para submissão de novas funcionalidades, correção de problemas ou melhorias para a *branch main* – a representação do atual estado do código em produção. A Figura 3.4 apresenta um exemplo de *Pull Request* criada pelos autores deste trabalho no *GitHub* no repositório do *frontend* do projeto.

Esta etapa do processo de desenvolvimento, chamada de *Code Review* ou de revisão de

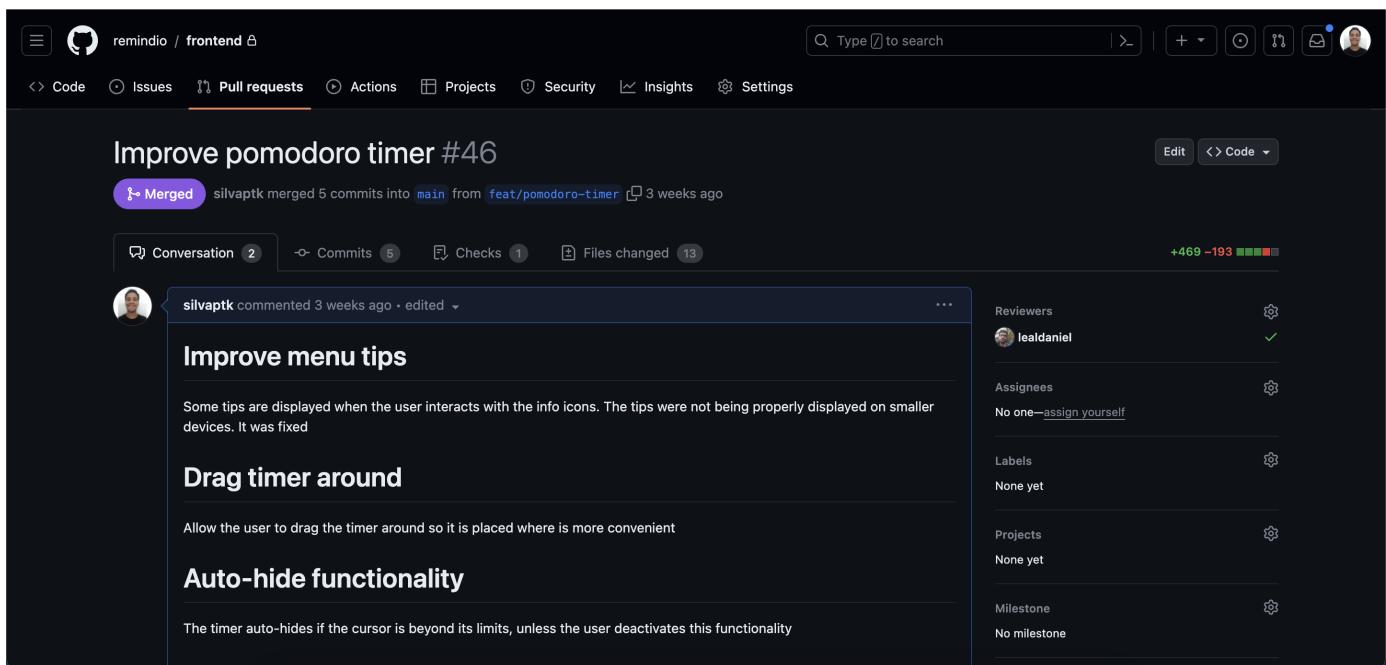


Figura 3.4: Exemplo de Pull Request do GitHub

código, é uma prática bastante sólida e importante pois garante que toda nova alteração passe pelo crivo de outras pessoas. Essa revisão impacta positivamente a qualidade de código submetido, por exemplo, ao prevenir a introdução inesperada de novos problemas, além de possibilitar que todos os membros de uma equipe de desenvolvimento se familiarizem de maneira contínua com novos trechos de código, facilitando a posterior manutenção do mesmo. A [Figura 3.5](#) demonstra uma discussão gerada a partir do processo de revisão de código:

Entrega e integração contínua

Após a etapa de revisão de código, o processo de Integração Contínua/Entrega Contínua (CI/CD) entra em ação. Essa prática, que é parte essencial do processo de desenvolvimento, do ponto de vista da cultura DevOps, garante que as alterações no código sejam testadas e, em seguida, unidas à *branch* principal de maneira eficiente e confiável. Após essa união, a entrega contínua entra em ação e atualiza a versão do projeto em produção ao enviar a versão mais recente à plataforma de hospedagem. Dessa forma, é possível garantir que a versão em produção esteja sempre atualizada. A [Figura 3.6](#) demonstra o fluxo completo do ciclo de desenvolvimento.

A CI/CD possui vantagens significativas. Por exemplo, ela permite a detecção de problemas em um estágio inicial, o que reduz o risco de falhas na produção. Além disso, a automação envolvida nesse processo aumenta a produtividade da equipe de desenvolvimento, permitindo que eles se concentrem em tarefas mais complexas enquanto o sistema cuida do processo de teste e implantação. Como afirmou o especialista em DevOps, Jez Humble, “a entrega contínua permite que você obtenha feedback rápido de seus usuários para que possa aprender com eles” ([HUMBLE e FARLEY, 2010](#)).

3.3 | CICLO DE DESENVOLVIMENTO

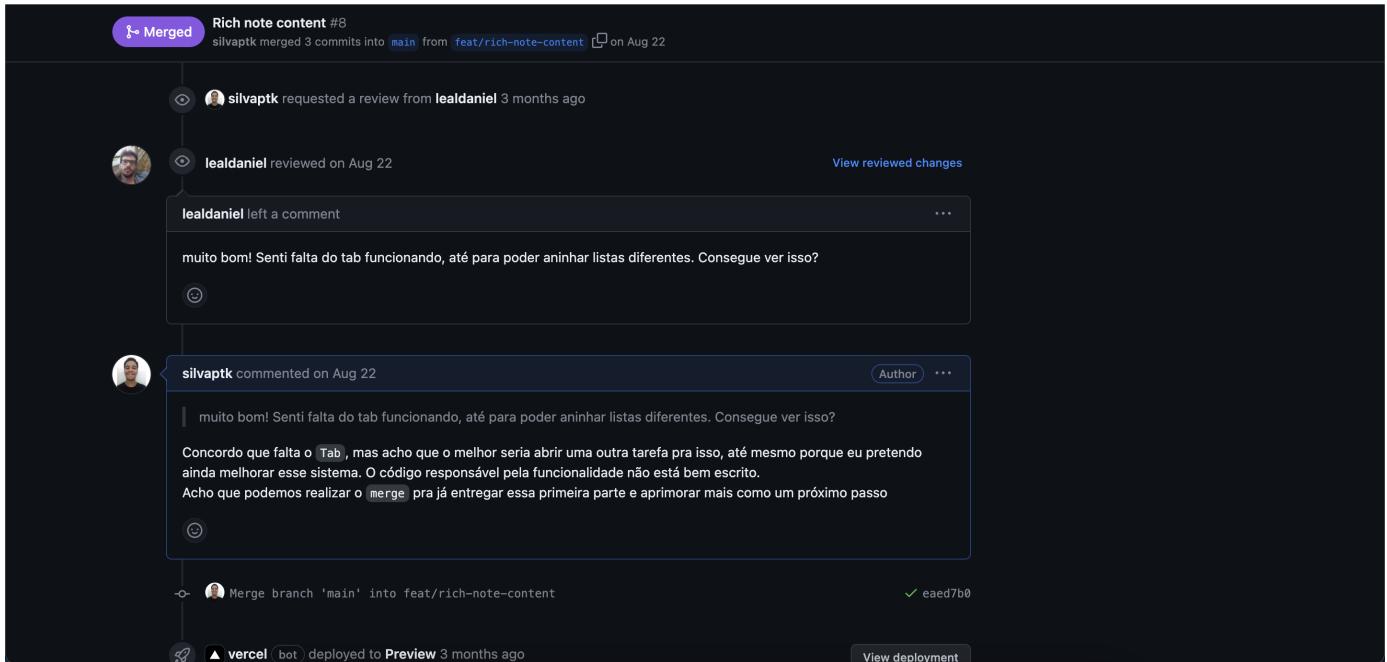


Figura 3.5: Exemplo de discussão produzida por revisão de código

Finalmente, o projeto se beneficia da CI/CD porque possibilita a entrega contínua de pequenas atualizações. Isso permite que a equipe de desenvolvimento responda de forma mais rápida e eficaz às mudanças, melhorando a qualidade do produto final e a satisfação do cliente. Finalmente, essa prática apoia a cultura de aprendizado e adaptação contínua, tornando o processo de desenvolvimento mais ágil e eficiente. Portanto, a integração contínua e a entrega contínua são componentes cruciais do ciclo de desenvolvimento do projeto, contribuindo significativamente para a qualidade do código e para a eficiência geral do processo.

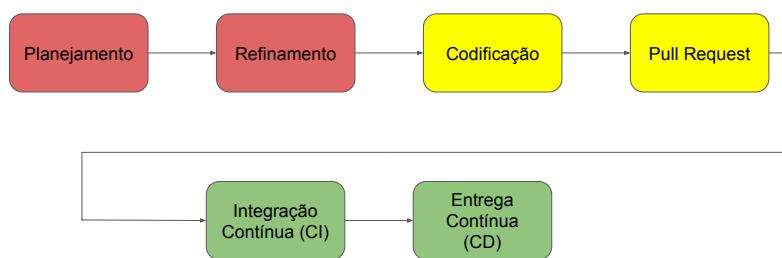


Figura 3.6: Ciclo de desenvolvimento completo

A sessão a seguir apresenta os frutos colhidos a partir da realização do ciclo de desenvolvimento descrito anteriormente ao apresentar as funcionalidades desenvolvidas, as

lições aprendidas e os resultados obtidos durante o período.

Capítulo 4

Resultados

Neste capítulo serão apresentados os resultados obtidos em relação ao desenvolvimento do projeto e as lições aprendidas ao longo do processo. Importante pontuar, inicialmente, que apesar dos pontos levantados e discutidos nos capítulos anteriores, o processo de um produto é algo dinâmico e heterogêneo, já que envolve diversos atores, interesses e opiniões. Em virtude disso, foi necessário por parte dos autores adaptabilidade para reagir e tomar as melhores decisões, com as informações existentes no momento, alinhados fortemente a mentalidade ágil que preza pela interação entre as pessoas à rigidez de processos.

4.1 Funcionalidades desenvolvidas

No contexto deste trabalho de conclusão de curso, as seguintes funcionalidades haviam sido implementadas no projeto, em ordem de entrega, conforme as subseções a seguir.

4.1.1 Interação com notas

O coração do Remindio estão nas notas. Elas constituem a parte central da plataforma e é onde os usuários mais interagem. Com isso em mente, grande parte do esforço no processo de concepção e desenvolvimento da plataforma se deu em prover às notas funcionalidades únicas, que garantissem vantagens competitivas à plataforma, como pode ser visto na [Figura 4.1](#). Deste modo, um usuário consegue:

- Criar notas
- Editar o conteúdo das notas
- Redimensionar o tamanho
- Minimizar e maximizar

Além disso, o conteúdo das notas contam com um editor Markdown, dando ainda mais poder para que o usuário consiga criar textos, anotações e listas de forma rica, flexível e personalizável.



Figura 4.1: Imagem mostrando um ambiente com diversas notas

Um ponto importante nessa funcionalidade foram os desafios técnicos enfrentados para prover ao usuário uma experiência que não desapontasse em diversas telas. Como as notas podem ser movimentadas, uma preocupação era garantir que estas não fossem perdidas caso o usuário mudasse o tamanho de sua tela, isso implicou em uma solução simples mas sofisticada de proporcionalidade, isso é, o tamanho das notas e sua localização são proporcionais ao tamanho da tela, permitindo que nenhuma informação seja perdida e seja possível interagir com as notas independente do dispositivo, como pode ser visto na Figura 4.2.

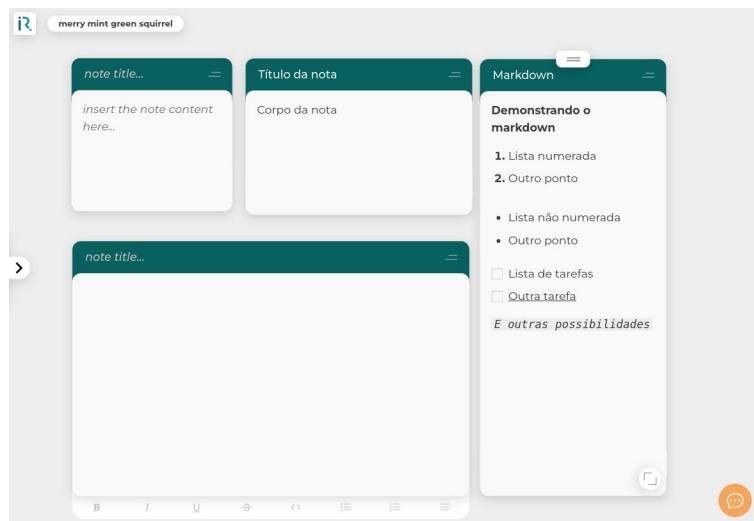


Figura 4.2: Imagem do mesmo ambiente, demonstrando a proporcionalidade das notas em uma tela menor

Essa solução demonstra que por trás da funcionalidade em si, existem diversos desafios a serem solucionados afim de garantir uma boa experiência aos usuários e foi através de testes após a entrega da funcionalidade que esse problema se mostrou, reforçando o argumento feito no Capítulo 3, sobre a importância de se trabalhar com sistema de *software*

em produção, pois foi a entrega rápida que permitiu a pronta identificação desse problema que finalmente possibilitou a implementação da solução definitiva, que acabou sendo útil também para atender os dispositivos móveis.

4.1.2 Autenticação

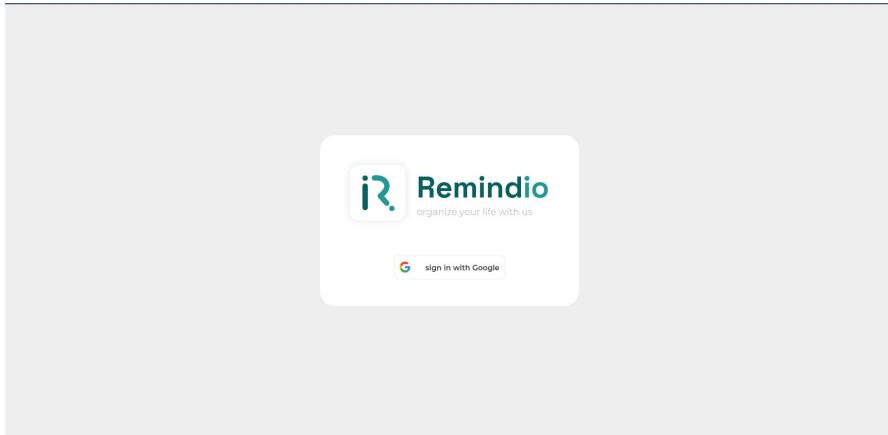


Figura 4.3: Tela de registro e login da plataforma

Através dessa tela, ilustrada na [Figura 4.3](#) um usuário consegue utilizar sua conta do Google para se registrar e utilizar a plataforma. Isso permitiu que as informações de um usuário pudessem ser conectadas a sua conta, assim, um usuário poderia recuperar todas as suas informações independente do dispositivo que estiver utilizando para acessar a plataforma.

Essa funcionalidade, apesar de fundamental, foi entregue posteriormente ao MVP do projeto, especificado na [Figura 3.1](#), em que, o usuário tinha um ambiente fixo e poderia interagir com as notas. Nesta versão, parte das informações estavam armazenadas no navegador, afim de garantir que já fosse possível testar a ferramenta no ambiente de produção.

4.1.3 Formulário de feedbacks

O *feedback* do usuário é um componente vital no desenvolvimento de todo produto, pois age como guia para melhorias em funcionalidades existentes e até sugestões de novas. Um dos objetivos desse projeto era trabalhar com sistema de *software* em produção justamente para desfrutar ao máximo desse importante aspecto. Com isso em mente, uma das *features* implementadas foi a possibilidade do usuário enviar sua opinião sobre a ferramenta através de um formulário contendo:

- Uma nota de 1 a 10
- Um espaço de texto para sugerir melhorias e realizar elogios e/ou críticas

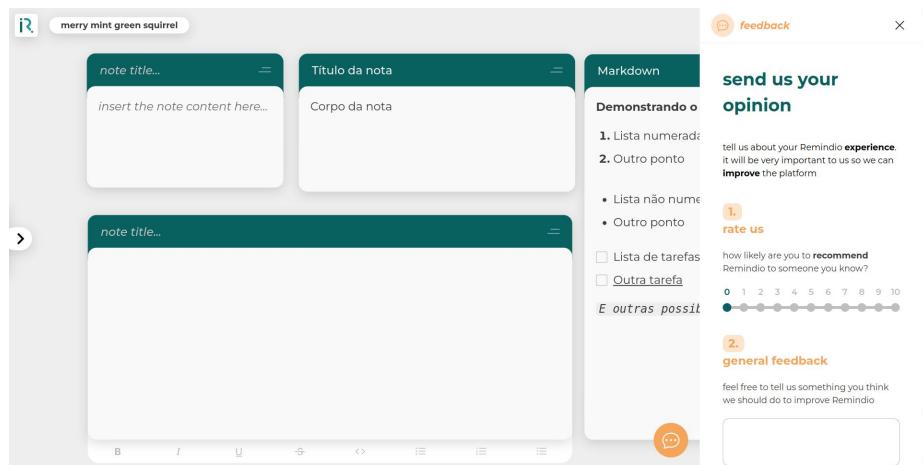


Figura 4.4: Captura de tela mostrando a aba de feedback que aparece ao clicar no ícone no canto inferior direito

A Figura 4.4 ilustra o formulário exibido como um menu lateral direito. Através desse formulário, os autores deste trabalho puderam ter acesso às percepções dos usuários e assim discutir a priorização das melhorias sugeridas.

4.1.4 Multi-ambientes

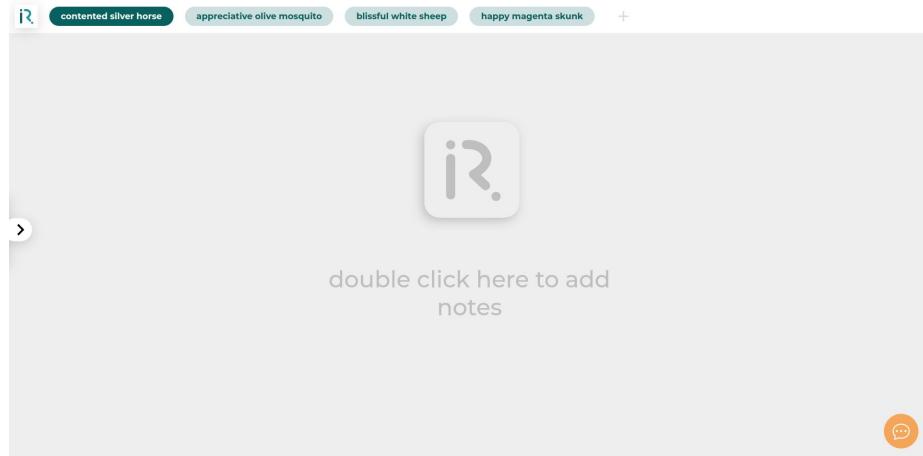


Figura 4.5: Imagem demonstrando um usuário com diversas ambientes na parte superior

A funcionalidade de multi-ambientes foi implementada com o objetivo de permitir que o usuário criasse vários ambientes, os espaços que contem as notas. Na primeira versão do projeto um usuário só poderia ter um ambiente e com a implementação da autenticação, essa funcionalidade ficou cada vez mais perto. A priorização dessa *feature* se deu a partir de *feedbacks* das pessoas que já estavam utilizando a plataforma, informando que seria interessante ter mais espaço para suas notas para poderem separar os diferentes contextos de suas vidas. A partir dessa implementação, um usuário consegue:

- Criar ambientes
- Editar nome dos ambientes criados
- Deletar ambientes criados

A implementação é apresentada na figura Figura 4.5 em forma de menu superior. A possibilidade de ter vários ambientes fez a plataforma passar a atender de maneira mais robusta o que se propunha a fazer: possibilitar aos usuários organizar os diversos contextos de sua vida, de maneira flexível, de acordo com suas necessidades e gostos.

4.1.5 Temporizador pomodoro

Um dos objetivos do Remindio é abordar todos os aspectos da organização de um indivíduo e unificar isso em uma única ferramenta. Nesse contexto, a gestão do tempo é de extrema importância e foi pensando nisso que uma das principais features desenvolvidas foi o temporizador Pomodoro. A técnica consiste em dividir o trabalho em seções de tempo, como 25 minutos, separados por breves intervalos, baseado na ideia de que pausas frequentes ajudam no foco, na atitude do usuário e a reduzir desperdício de tempo (BANIQUED e ARISTON, 2019).

O temporizador foi implementando como uma funcionalidade desacoplada dos ambientes, visando permitir que o usuário usufrua da mesma de forma paralela. Nesse ponto do projeto, o usuário é capaz de organizar as informações através de ambientes e de notas e de organizar seu tempo através do temporizador. Além disso, apesar de ainda não ter sido implementada, se visualiza a funcionalidade de agenda nesse mesmo nível do *timer* – uma funcionalidade além dos ambientes.

Dessa forma, um usuário consegue:

- Pausar, iniciar e reiniciar o temporizador
- Editar configurações de tempo e quantidade de ciclos
- Mudar a posição do temporizador

A Figura 4.6 é uma captura de tela que demonstra o funcionamento do temporizador.

4.1.6 Responsividade para dispositivos móveis

Uma preocupação desde o início do desenvolvimento do projeto era como levar o Remindio para *smartphones*, por conta da sua proposta de interação diferenciada. Qualquer ferramenta hoje possui uma alternativa *mobile*, já que os dispositivos móveis desempenham um papel importante nas nossas vidas. Com isso em mente, permitir que os usuários pudessem usufruir das funcionalidades da plataforma enquanto não estivessem perto de seu principal dispositivo de trabalho ou estudo era essencial para garantir a adoção da ferramenta.

Inicialmente, tinha-se a percepção de que não seria possível replicar a interatividade da plataforma nos aparelhos móveis, porém, após alguns testes e ajustes conseguiu-se chegar

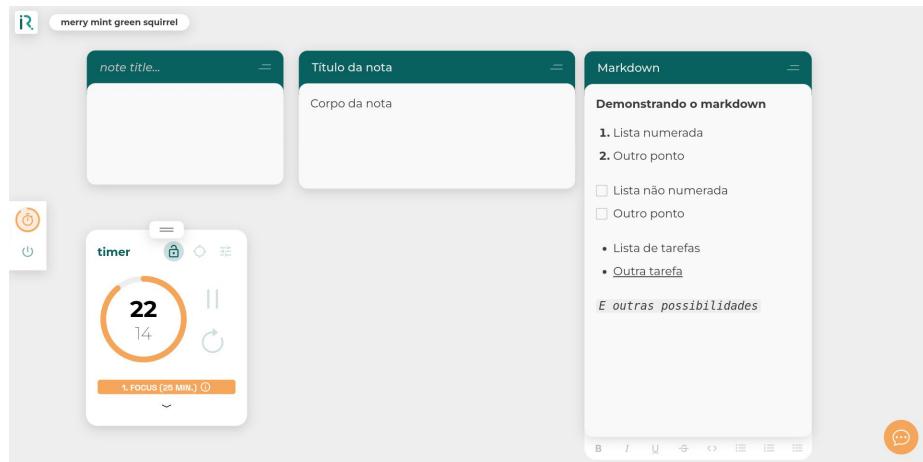


Figura 4.6: Captura de tela demonstrando o temporizador Pomodoro ao lado do menu de lateral onde pode ser acessado

à uma solução em que a essência da ferramenta não fosse perdida. Com isso, os usuários conseguem acessar e utilizar todos os recursos da plataforma de maneira similar, através do navegador de seus *smartphones*.

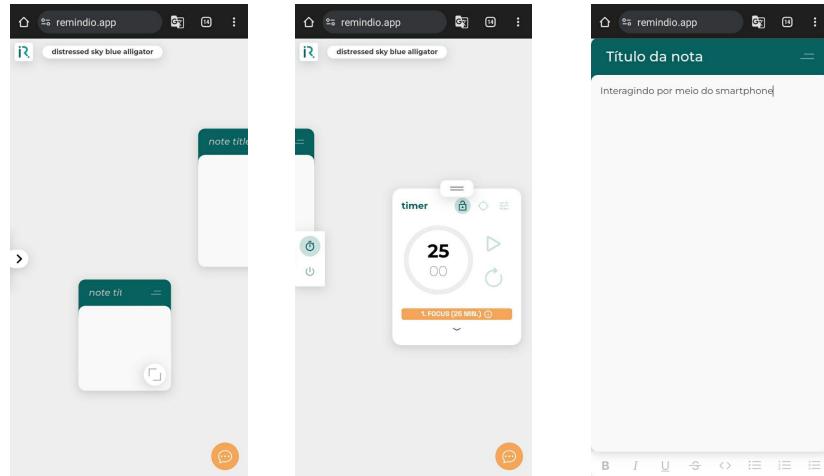


Figura 4.7: Capturas de tela mostrando o funcionamento da plataforma em um dispositivo móvel

Como pode ser visto na [Figura 4.7](#), ao utilizar o Remindio através de um dispositivo móvel, através do sistema de proporcionalidade – citado na [Subseção 4.1.1](#) – as notas são redimensionadas para encaixar verticalmente na tela, assim, o usuário consegue através do movimento de arrastar, percorrer horizontalmente o seu ambiente. Ao encontrar uma nota que deseja editar e tocá-la a nota é maximizada automaticamente permitindo a edição de seu título e conteúdo.

4.1.7 Compartilhar ambientes em modo de leitura

A última funcionalidade implementada para a plataforma foi a possibilidade de um usuário compartilhar seu ambiente com outros usuários, para isso duas maneiras de realizar esse compartilhamento foram implementadas:

- Compartilhar por meio de um *link*
- Compartilhar diretamente através do e-mail do outro usuário

O menu lateral que representa a funcionalidade pode ser visto na [Figura 4.8](#), do lado direito.

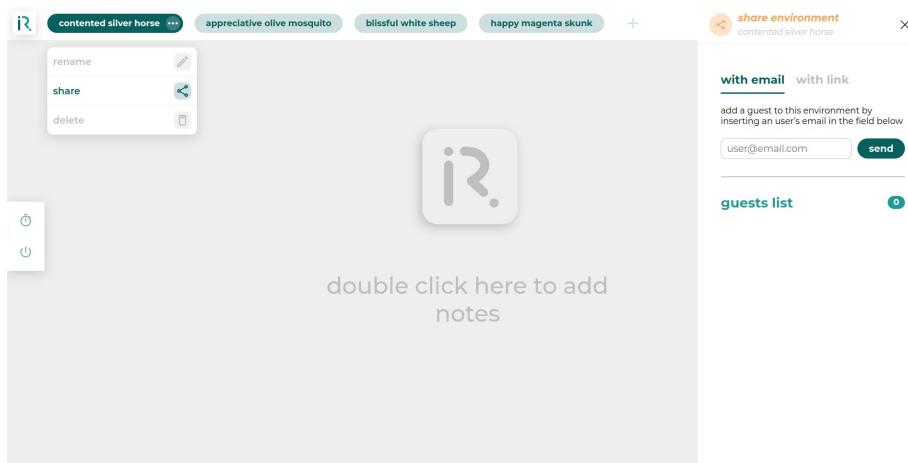


Figura 4.8: Captura de tela demonstrando o menu de compartilhamento de ambientes

No primeiro momento, os usuários convidados só podem visualizar o ambiente, deixando a possibilidade de permitir que estes também interajam com as notas para o futuro. Essa decisão foi tomada devido aos desafios que a utilização concorrente traria, já que para garantir uma experiência aceitável precisaria ser implementada uma comunicação inversa à tradicional, ou seja, o servidor precisaria acionar o cliente para que este refletisse para um usuário as alterações realizadas por um outro naquele mesmo ambiente. Com esse entendimento, foi decidido permitir somente o modo leitura a usuários convidados.

4.2 Percepções e resultados

Até o final deste trabalho, o Remindio possuía 31 contas, 121 ambientes e 211 notas criadas. Com a divulgação para essas pessoas sendo feita majoritariamente de forma pessoal, isso é, em conversas em que podia-se colher as impressões da mesma. De maneira geral, as percepções sobre a plataforma foram positivas. Os autores notaram que as pessoas ficavam impressionadas com a plataforma, gostavam dela e a adotavam, ou a achavam visualmente agradável e interessante, mas não ficavam suficientemente impressionadas para adotá-la no cotidiano. Essas reações conversam com o discutido no [Capítulo 1](#), sobre a influência do gosto pessoal para a adoção desse tipo de ferramenta. Em contrapartida,

essas reações à plataforma reforçam a hipótese levantada de que o Remindio possui um *design* único, o que representa uma vantagem competitiva em relação às demais.

Alguns usuários utilizaram o formulário de *feedback* desenvolvido para compartilhar sugestões e suas percepções. Um usuário, que atribuiu nota 9 na pergunta se recomendaria a plataforma a outra pessoa, sugeriu que o menu de opções das notas fosse exibido mais rapidamente. Essa sugestão foi prontamente acatada e implementada em uma das entregas de melhorias da plataforma. Outro usuário, também com nota 9, expressou: “Adorei a plataforma. A criação de *cards* e gerenciamento de workspaces é bastante intuitiva, apesar de a interface exigir interações demais para realizar alguma ação. Ademais, é prático e fácil de utilizar e tudo funciona com a agilidade esperada.”. Finalmente, com uma nota mais baixa, 7, um utilizador da plataforma sugeriu facilitar a forma como novos ambientes são criados, bem como consertar um problema persistente que faz as notas parecerem embaçadas.

Além dos *feedbacks* feitos através do formulário, algumas percepções foram coletadas com as pessoas no momento em que se divulgava a plataforma a elas. Ao entrar na plataforma e se deparar com o ambiente tutorial e começar a explorar as funcionalidades um usuário sem querer deletou uma nota e ao não conseguir recuperá-la com o popular comando `Ctrl + Z`, sentiu falta dessa funcionalidade que na sua opinião é muito necessária. Além dessa ideia, o usuário achou que seria melhor se as notas pudessem ser movimentadas pelo seu cabeçalho, sem a necessidade do ícone superior que aparece quando o mouse fica por cima da nota. Já outros usuários não fizeram menção a forma como notas são repositionadas mas disseram que gostariam da possibilidade de poder customizar a cor do cabeçalho.

4.3 Lições aprendidas

Idealizar, implementar e manter um sistema é difícil, principalmente se somado a isso existe toda a complexidade de infraestrutura, observabilidade e cuidado ao se operar um sistema de *software* com usuários utilizando. Durante esse projeto, ambos os autores acabaram acumulando diversas funções que num contexto corporativo não seria comum. Dessa forma, esse acúmulo implica inevitavelmente na impossibilidade de realizar todas as ações que são tidas como essenciais na construção de um produto como realizar entrevistas mais formais com usuários afim de colher *feedbacks* mais qualitativos sobre a plataforma ou testes de usabilidade. Nesse contexto, criar e manter um sistema de *software* em produção, cuidando de todo o ciclo de desenvolvimento foi o principal aprendizado para os autores, pois foi algo diferente do que praticado durante a graduação, em que normalmente só é necessário implementar e testar aquilo que foi previamente especificado.

Além do aprendizado técnico, referente às tecnologias e arquiteturas empregadas, configurar as aplicações em produção foi uma experiência diferente por se tratar de algo que os autores não estavam muito acostumados a fazer. Essa experiência contribuiu com conhecimentos teóricos e práticos que estão presentes no final do ciclo de vida de um sistema de *software*, papel esse que muita vezes, no mundo corporativo, é desempenhado por um profissional específico.

A experiência de trabalhar com um sistema em produção e em uso contínuo pelos

usuários proporcionou aos autores uma oportunidade valiosa de atuar mais próximos aos clientes. Esta vivência enriquece as capacidades de pensamento crítico e estratégico, especialmente porque estar constantemente exposto a *feedbacks* e sugestões demanda uma reflexão cuidadosa sobre decisões e priorizações. Há um equilíbrio a ser mantido entre implementar melhorias nas experiências existentes, para não desapontar os usuários atuais, e desenvolver novas funcionalidades que agradem os usuários presentes e atraiam novos. Além disso, a habilidade de discernir e filtrar as percepções e sugestões dos usuários torna-se um desafio, particularmente em situações onde as necessidades ou desejos de um usuário podem não se alinhar com os de outros. Esta habilidade de navegar por demandas diversas e muitas vezes conflitantes é essencial para o sucesso de uma plataforma.

Capítulo 5

Conclusão

Em síntese, ao final deste trabalho, foi entregue uma plataforma de organização pessoal com um design único, disponível gratuitamente ao público. A plataforma pode ser acessada em: <https://remindio.app/>. Na construção dessa plataforma foram empregadas boas práticas de engenharia de *software*, utilizando as tecnologias e implementando as decisões arquiteturais discutidas ao longo da monografia.

Em relação aos aspectos não técnicos, os autores praticaram no dia a dia do desenvolvimento e concepção do Remindio as práticas abordadas no Capítulo 3. A equipe realizou as cerimônias que permitiam discutir estratégias, priorizações e melhorias necessárias em quase todas as semanas, exceto quando ambos concordavam em discutir progressos, problemas e pendências de maneira assíncrona, mais informal e rápida para dedicar mais tempos às demandas em andamento.

Das funcionalidades planejadas, apenas a agenda, que pretendia fornecer aos usuários mais suporte no aspecto da gestão de tempo, não foi implementada. Desde o início enxergava-se essa *feature* como interessante, já que traria mais poder de unificação à plataforma, porém outras funcionalidades foram priorizadas por conta da dificuldade de implementar uma agenda competitiva em relação aos concorrentes. Além disso, as funcionalidades priorizadas foram notadamente mais importantes para aceitação da plataforma - a funcionalidade de multi-ambientes, as notas e o temporizador Pomodoro.

Por fim, é importante destacar que, como uma plataforma dinâmica, o Remindio possui diversas possibilidades para inovações e melhorias. Além da implementação da funcionalidade de agenda, há espaço para aprimoramentos de usabilidade e para introdução de outras funcionalidades para complementar e enriquecer as existentes. Alguns exemplos são:

1. **personalização de cor do cabeçalho da nota:** sugestão de usuários que pode melhorar sua experiência e agregar valor à plataforma
2. **possibilidade de múltiplos usuários modificando mesmo ambiente:** representa uma evolução do compartilhamento de ambientes que foi implementado
3. **duplicação de ambientes:** permitir que um ambiente e as notas contidas sejam duplicadas

4. **mover notas entre ambientes:** possibilidade de copiar uma nota de um ambiente para outro

Essas possibilidades reforçam a natureza evolutiva do produto, possibilitando que a plataforma se mantenha relevante, útil e atrativa para quem já a utiliza e para novos possíveis usuários.

Referências

- [BANIQUED e ARISTON 2019] W. B. BANIQUED e C. D. ARISTON. “The pomodoro: effectiveness to grade 10 science students’ time management”. *QSU Research Journal* 8.1 (2019). ISSN: 2945-4921. URL: <https://www.ejournals.ph/article.php?id=17041> (citado na pg. 45).
- [BIERMAN *et al.* 2014] Gavin BIERMAN, Martín ABADI e Mads TORGERSEN. “Understanding typescript”. In: *ECOOP 2014 – Object-Oriented Programming*. Ed. por Richard JONES. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 257–281. ISBN: 978-3-662-44202-9 (citado na pg. 23).
- [BODUCH 2016] Adam BODUCH. *Flux architecture*. Packt Publishing Ltd, 2016 (citado na pg. 26).
- [BRANCO 2015] Marta BRANCO. ““Eu vou fazer, mas...”: Um Estudo Exploratório sobre Ansiedade-Estado, Preocupação e Procrastinação Académica em Estudantes Universitários”. Diss. de mestr. Lisboa, Portugal: Instituto Superior de Psicologia Aplicada, 2015 (citado na pg. 1).
- [BRITTON e TESSER 1991] Bruce K. BRITTON e Abraham TESSER. “Effects of time-management practices on college grades”. *Journal of Educational Psychology* 83.3 (1991), pp. 405–410. ISSN: 0080-6234. DOI: [10.1590/S0080-62342011000300025](https://doi.org/10.1590/S0080-62342011000300025) (citado na pg. 1).
- [A. W. BROWN 2000] Alan W. BROWN. *Large-Scale, Component Based Development*. USA: Prentice Hall PTR, 2000. ISBN: 013088720X (citado na pg. 25).
- [T. BROWN 2009] Tim BROWN. *Change by Design: How Design Thinking Transforms Organizations and Inspires Innovation*. HarperBusiness, 2009 (citado na pg. 34).
- [CARVALHO FARIAS *et al.* 2011] Sílvia Maria de CARVALHO FARIAS, Olga Lúcia de CARVALHO TEIXEIRA, Walter MOREIRA, Márcia Aparecida Ferreira de OLIVEIRA e Maria Odete PEREIRA. “Caracterização dos sintomas físicos de estresse na equipe de pronto atendimento”. *Revista da Escola de Enfermagem da USP* 45.3 (jun. de 2011), pp. 722–729. ISSN: 0080-6234. DOI: [10.1590/S0080-62342011000300025](https://doi.org/10.1590/S0080-62342011000300025) (citado na pg. 1).

- [H. CATLIN e M. L. CATLIN 2011] Hampton CATLIN e Michael Lintorn CATLIN. *Pragmatic Guide to Sass*. Pragmatic Bookshelf, 2011. ISBN: 1934356840 (citado na pg. 23).
- [COCKBURN 2004] Alistair COCKBURN. *Agile Software Development: The Cooperative Game*. Addison-Wesley Professional, 2004 (citado na pg. 33).
- [COCKBURN 2005] Alistair COCKBURN. *Hexagonal architecture*. 2005. URL: <https://alistair-cockburn.us/hexagonal-architecture/> (acesso em 19/09/2023) (citado nas pgs. 16, 17).
- [DUCKWORTH e SELIGMAN 2005] Angela L. DUCKWORTH e Martin E.P. SELIGMAN. “Self-discipline outdoes iq in predicting academic performance of adolescents”. *Psychological Science* 16.12 (2005), pp. 939–944. DOI: [10.1111/j.1467-9280.2005.01641.x](https://doi.org/10.1111/j.1467-9280.2005.01641.x) (citado na pg. 1).
- [FOWLER 2015] Martin FOWLER. *MonolithFirst*. 2015. URL: <https://martinfowler.com/bliki/MonolithFirst.html> (acesso em 04/12/2023) (citado na pg. 15).
- [GAMMA *et al.* 1994] Erich GAMMA, Richard HELM, Ralph JOHNSON e John VLISSIDES. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994 (citado na pg. 17).
- [HAERDER e REUTER 1983] Theo HAERDER e Andreas REUTER. “Principles of transaction-oriented database recovery”. *ACM Comput. Surv.* 15.4 (dez. de 1983), pp. 287–317. ISSN: 0360-0300. DOI: [10.1145/289.291](https://doi.org/10.1145/289.291). URL: <https://doi.org/10.1145/289.291> (citado na pg. 22).
- [HUMBLE e FARLEY 2010] Jez HUMBLE e David FARLEY. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010 (citado nas pgs. 36, 38).
- [JOCA *et al.* 2003] Sâmia Regiane Lourenço JOCA, Cláudia Maria PADOVAN e Francisco Silveira GUIMARÃES. “Estresse, depressão e hipocampo”. *Brazilian Journal of Psychiatry* 25 (dez. de 2003), pp. 46–51. ISSN: 1516-4446. DOI: [10.1590/S1516-44462003000600011](https://doi.org/10.1590/S1516-44462003000600011). URL: <https://doi.org/10.1590/S1516-44462003000600011> (citado na pg. 1).
- [Robert C MARTIN 2012] Robert C MARTIN. *NO DB*. 2012. URL: <https://blog.cleancoder.com/uncle-bob/2012/05/15/NODB.html> (acesso em 25/09/2023) (citado na pg. 21).
- [Robert Cecil MARTIN 2017] Robert Cecil MARTIN. *Clean Architecture: A Craftsman’s Guide to Software Structure and Design*. Pearson, 2017 (citado na pg. 16).
- [NIELSEN 1994] Jakob NIELSEN. *Usability engineering*. Elsevier, 1994 (citado nas pgs. 11, 34).

REFERÊNCIAS

- [PAVLENKO *et al.* 2020] Andrei PAVLENKO, Nursultan ASKARBEKULY, Swati MEGHA e Manuel MAZZARA. “Micro-frontends: application of microservices to web front-ends” (mai. de 2020). DOI: [10.22667/JISIS.2020.05.31.049](https://doi.org/10.22667/JISIS.2020.05.31.049) (citado na pg. 22).
- [RIES 2011] Eric RIES. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business, 2011 (citado na pg. 34).
- [TIDWELL 2010] Jenifer TIDWELL. *Designing Interfaces*. O'Reilly Media, Inc., 2010. ISBN: 1449379702 (citado na pg. 31).